ESPRIT

COMMISSION OF THE EUROPEAN COMMUNITIES

Directorate-General
TELECOMMUNICATIONS, INFORMATION
INDUSTRIES AND INNOVATION

ESPRIT '87 Achievements and Impact

Part 1

North-Holland

ESPRIT '87

Achievements and Impact

ESPRIT '87

Achievements and Impact

Proceedings of the 4th Annual ESPRIT Conference
Brussels, September 28-29, 1987

Edited by

COMMISSION OF THE EUROPEAN COMMUNITIES
Directorate-General TELECOMMUNICATIONS,
INFORMATION INDUSTRIES and INNOVATION

Part 1

1987

NORTH-HOLLAND
AMSTERDAM • NEW YORK • OXFORD • TOKYO

ISBN Part 1: 0 444 70331 4
ISBN Part 2: 0 444 70332 2
ISBN Set : 0 444 70333 0

Publishers:
ELSEVIER SCIENCE PUBLISHERS B.V.
P.O. Box 1991
1000 BZ Amsterdam
The Netherlands

Sole distributors for the U.S.A. and Canada:
ELSEVIER SCIENCE PUBLISHING COMPANY, INC.
52 Vanderbilt Avenue
New York, N.Y. 10017
U.S.A.

Publication No. EUR 11192 of the
Commission of the European Communities,
Directorate-General Telecommunications,
Information Industries and Innovation,
Luxembourg

PRINTED IN THE NETHERLANDS

FOREWORD

The 4th ESPRIT Conference will be held in the Palais des Congrès in Brussels from the 28th to 30th September 1987. Well over 1,000 participants are expected to attend about 120 presentations, and develop contacts with colleagues in their own and other disciplines. Some 50 of the ESPRIT projects are planning to demonstrate over one hundred computer systems, to highlight the achievements of ESPRIT during the past twelve months. Bearing in mind that we are only one third of the way into this ten year programme, the scale of the event is especially gratifying.

The first two days are traditionally devoted to presentations, on the *achievements and impact* of selected ESPRIT projects, structured into plenary and parallel sessions, with a majority of papers read in sessions that cover more than one of the ESPRIT action areas : Microelectronics, Software Technology, Advanced Information Processing, Office Systems and Computer Integrated Manufacturing, as well as one sesssion of papers on the network development projects of ESPRIT.

On the third day of the Conference, the Information Technology Forum Day, industrial and political leaders will discuss the theme *Europe 1992 : Technology and Market* in the morning, while the afternoon is devoted to the presentation of the second phase of ESPRIT. The Conference is a good opportunity for the participants to get a first-hand report on the substantial work going on in ESPRIT, and on the many results already obtained and now being exploited. The participants also take advantage of the opportunity to meet each other and discuss items of mutual interest, this being particularly useful because of the good balance between the various specialised areas and also across different European national boundaries. This closer relationship will reap greater benefits as we look to the second phase of ESPRIT.

I would especially like to thank the contributors whose presentations are essential to the overall success of the Conference, as well as the project teams who set up the demonstrations, and thank the various organisations who have made equipment available to them.

J.M. Cadiou
Director
Information Technologies - ESPRIT

ACKNOWLEDGEMENTS

The Commission of the European Communities thanks the following members of the conference programme committee and reviewers for their contribution to the 4th ESPRIT Conference.

André, E.
Antreich
Arrowsmith, P.
Baldi, L.
Balk, P.
Balter, R.
Bardsley, W.
Barrow, C.
Bensahel, D.
Bentley, M.
Bjoern-Andersen, N.
Coeure, P.
Dato, M.
Dirks, H.
Doumeingts, G.
Drechsel, G.
Eggermont, L.D.J.
Encarnacao, J.L.
Evans, R.
Haton, J.P.
Heckl, H.
Hemment, P.L.F.
Heuberger, A.
Hezel, R.
Holt, W.
Jacquart, R.
Kirchoff, U.
Koch, G.
Leproni, L.
Levi, G.
Littlewood, B.
Madsen, O.B.
Maes, H.
Maréchal, G.
Mertens, R.
Meyer, W.
Morgan, D.
Murchio, P.
O'Brien, C.
O'Mathua, C.
Parker, G.A.
Prini, G.
Ricco, B.
Robertson, J.
Schneider, B.
Schneider, B.
Schulz, M.J.
Shorter, D.
Tedd, M.
Thorp, T.L.
Treleaven, P.
Trullemans, C.
Turini, F.
Van de Wiele
Van der Wolf, P.
Vassiliou, Y.
Vissers, C.A.
Williams, E.W.
Wittig, T.
Young, R.W.
Zyss, J.

CONTENTS
Part 1

I. MICROELECTRONICS

1. Process Overview

2. Basic Technologies

3. Application Technologies

4. High Level Systems Design

5. CAD for VLSI Design

II. SOFTWARE TECHNOLOGY

1. Environments

2. Advanced Environments

3. Metrication and Management

4. Formal Methods

III. ADVANCED INFORMATION PROCESSING

1. Knowledge Engineering

2. Systems Architecture and Design

3. Signal Processing, Natural Languages

4. Expert Systems

5. Documents Architecture, Storage and Retrieval (part 1)

I. MICROELECTRONICS

Paper Presented in the Plenary Sessions:

Parallel Sessions

Project No. 412

BIPOLAR CMOS ESPRIT PROJECT

P.A.H. Hart (Philips), A. Wieder (Siemens)

Introduction

In this paper will be described the BICMOS project 412.
First of all, it should be stated that the objective of the project is to combine bipolar IC processing and CMOS processing into a combined bipolar CMOS process. This process should be usefull for information processing and marry in an economic way the advantages of bipolar and CMOS. The advantage of bipolar processing is particularly high gain and high speed, those of CMOS processing low dissipation, the ease to build complex circuitry and high process yield. This renders it possible to make large circuits possessing "bipolarlike" speeds and CMOS like complexity and low dissipation. The general aim is enhancement of performance. In some cases a reduction of circuit complexity of 30-40% was observed (1) while (cf. 2,3) a reduction of circuit size by approximately one half was observed in comparison with a bipolar--only circuit.

It should be mentioned that particularly in USA and Far East there is a great number of laboratories and companies engaged. Such companies are DEC, Honeywell, Hughes, Motorola, National, Texas instruments, RCA, Hitachi, NEC, Ricoh, Toshiba, Samsung etc. Not all these companies aim at the same application but all combine bipolar and CMOS in one process. The majority of companies places great emphasis on applications like high performance memories and high performance gate arrays or similar structures. The other most important area is analog circuits having a (large) digital content.
In this paper first some general statements will be made to outline the project, then the nature of the work and some results will be described.

The project

The BICMOS 412 ESPRIT project is carried out by Philips and Siemens as the main partners. Philips is the main contractor. The Universities of Dublin and Stuttgart are subcontractor to Philips and Siemens respectively.
An integrated circuit technology combining bipolar and CMOS technology is being developed at Philips and Siemens.
Additionally extensive physical and electronic studies and design are being undertaken in support of the technological work. In the matter of electronics design Philips and Siemens are being assisted by their respective subcontractors. The project has now entered its third year (started 1-4-1985). Over a five year period, Philips expects to deliver 150 manyears and Siemens 50 manyears of effort.

Because of their respective product backgrounds Philips and Siemens apply somewhat different emphasis. The emphasis by Siemens is on digital systems, that one at Philips is analog systems with a digital content. This gives their cooperation an interesting aspect. In order to have full benefit from the inhouse design culture and CAD facilitities the, compulsory, starting point has been a standard CMOS process at both Philips and Siemens. The work then essentially consists in the addition of the bipolar part. As stated Philips and Siemens have somewhat different interest and histories, the way in which their cooperation is effected is not unimportant. This difference in background, however, makes the project more interesting, less prone to duplication and it certainly leads to a deeper mutual understanding. The basic principle of work in each company is that we exchange process knowledge, agree on a common set of design rules for test vehicles, design circuits in each other's processes and then exchange these test circuits to do a common evaluation. Subordinate to all this is an exchange of modelling knowledge and measuring techniques.
Present CMOS and bipolar processes both comprise approximately 12 mask steps. A naïf addition of both processes would double the price of the joint process and render it economically unattractive. The processes therefore have to be merged in such a way that the result is economically viable and still retains the advantages of both the bipolar and the CMOS discipline.
A scheme to combine N-well CMOS with the bipolar process is outlined in fig. 1a.

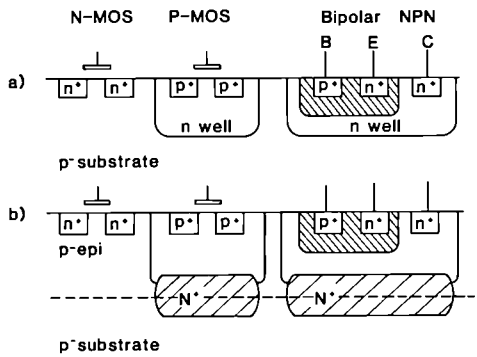


Fig. 1 Merging of bipolar to a N-Well CMOS process

The N-well CMOS is shown at the left. A vertical NPN bipolar transistor is visible on the right. For the bipolar transistors use is made of the same diffusions as are already used for the CMOS part except for one additional P region (hatched). In principle, therefore, only one additional mask is needed. In practice, (cf fig. 1b) it is preferable in the Philips and Siemens processes to use an epitaxial layer and additional N buried layers. This prevents latch up and greatly enhances the performance of the NPN transistor. The latter points were investigated by Siemens at the start of the project. They found that the collector series resistance dropped from 200 ohms to 30 ohms and that the current amplification of a (parasitic) pnp in a latch-up circuit dropped from 40 to 1, hereby greatly enhancing freedom from latch-up.

The objectives.

The present plan provides for a bipolar CMOS process-development which will allow a 200 K transistor complexity chip to be made and available at the end of year five. This challenging task has been divided up as follows.

- In the first three years a process named "BICMOS I" is to be developed by Philips and by Siemens. By Philips the feasibility of the process is to be demonstrated by a circuit of 20 K transistor complexity level. The Siemens process should have a bipolar transit frequency of 3 GHz and a CMOS toggle frequency of 500 MHz. Furthermore basic circuits are to be investigated and optimized, this will include an investigation of novel circuits now possible with a free mix of MOS and bipolar circuitry.
- In the second half of the first three years a process "BICMOS II" is to be designed. This process then should be capable of yielding the 200 K demonstrator at Philips. At Siemens a toggle frequency of 1200 MHz and a bipolar transitfrequency of 9 GHz is expected to be achieved.

The implementation of the work

The first year was largely spent determining the proper requirements for the process and the circuits, and doing all the experiments needed to confirm views on technology and substantiate the many simulations done in the first year.

In the first year, too, a technology called BICMOS OP using 2/um lithography, which had been developed previously to this Esprit project by Philips(2), was used to give a "flying" start with testing of electronic ideas and for calibration of the process simulation tools. Both Philips and Siemens had experimental circuits on these test masks (3). The floorplan has been described in (3); for convenience the list of devices and circuits is reproduced here in Table I, while some new device parameters of BICMOS 0P and BICMOS 1P are presented in Table II. A more detailed cross section of BICMOS 1P is shown in fig. 2.

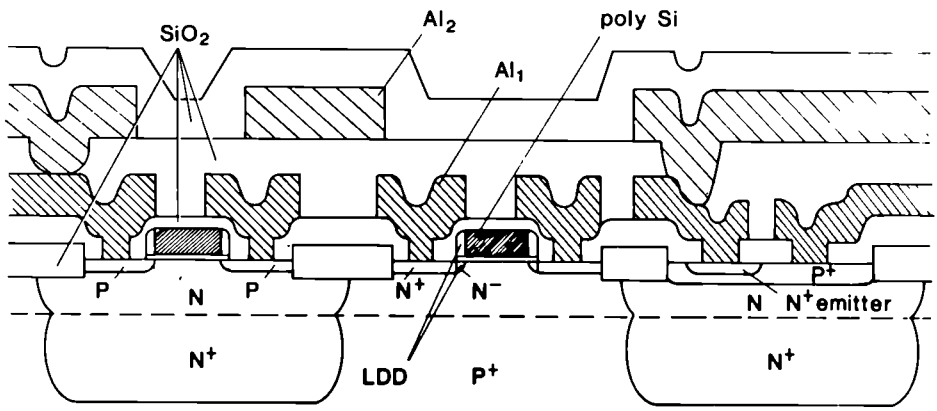


Fig. 2 Cross section of BICMOS 1P. The CMOS part also describes that of BICMOS 2P and BICMOS 2S.

TABLE I

Test structures and circuits on BICMOS test masks.

Philips	Siemens
Components for characterisation	Latch up investigation
Full adder chain with BICMOS buffers	BICMOS driver for ECL
	C-MOS driver for ECL
Bipolar and C-MOS shift registers	Philips process evaluation modules Gilbert cell
Siemens operational amplifiers	BICMOS operational amplifiers
Gyrator and other filter components	MOS differential pairs
2 K RAM	ECL devider static and dynamic
Yield modules for process steps	MOS arrays, matching structure
Siemens optical dimension control	Bipolar differential pairs
I 2C transceiver	
Balaced resonater VEO	
Sigma Delta Modulator	
ROM	

TABLE II

		BICMOS 0 P	BICMOS 1P NW	NWS
NPN	HFC	200	105	100
	FT	4.2 GHz	6 GHz	4GHz
	Vearl	75 V	23	90 V
	BVCEO	16 V	8V	12 V
PNP	Wb	3.5 µm	2.0 µm	2.0 µm
	Hfe	100	46	126
	FT	30 MHz	not yet measured	
	Vearly	25 V	40 V	15 V
	BVCEO	20 V	14 V	17 V
NMOS	Vt	0.75 V	0.9 V	-
	K´	0.44 V 1/2	0.50 V 1/2	-
	ß	50 µ A/V2	77 µA/V2	-
PMOS	Vt	-0.80 V	-1.1 V	-
	k	0.76 V 1/2	0.65 V 1/2	-
	ß	17µA/V2	23 µA/V2	-

At Siemens, basic experiments with a BICMOS 0S technology were performed on the basis of an existing 1.4/um N-well CMOS process using a test chip available at the time, (4). The results were used together with MOS, bipolar and BICMOS circuit simulation as the basis for the design of basic BICMOS circuits. Fig. 3 exemplifies the result of a basic study by Siemens.

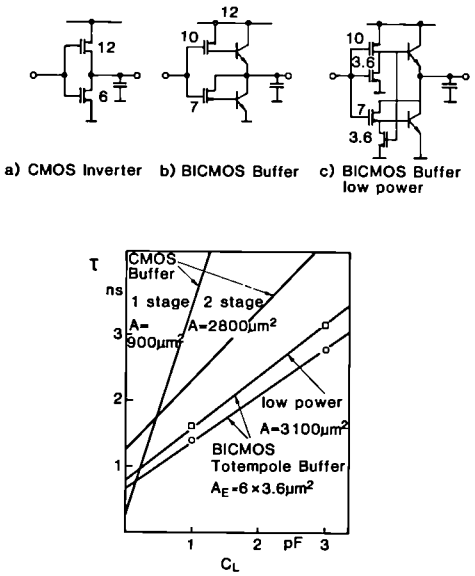


Fig. 3 Circuit schematics of CMOS and BICMOS buffers. Buffer delay versus load capacitance.

The University of Dublin assisting Philips with electronic design designed circuits on a BICMOS 0P test mask. Their objective is an FIR filter having a challenging clock rate of 50 MHz.

The University of Stuttgart, supporting Siemens with circuit optimisation, tooled up for this task in the first year. In order to cut down CPU time a mixed strategy of stochastic and deterministic search procedures for design centering was developed there. A criterion to automatically switching between the two strategies has already been implemented in the program package.

In the second year the results of BICMOS 0 were sorted out and a roughly similar operation started up, but now in the BICMOS 1, process; a cross section is shown in fig. 2.
The circuits involved comprise digital and analog functions using bipolar and CMOS both separate and in "arbitarily" mixed form, cf. Table I and, for BICMOS 1P parameters, cf. Table II.

The point of departure for the research on the BICMOS 1 process was a standard N well process,namely 1.5 and 1.4 um in Philips and Siemens respectively. Philips used a LDD structure in the NMOS and buried P and N layers whereas Siemens worked with $TaSi_2$ gate and buried N layers. This of course resulted in somewhat different BICMOS processes. Nevertheless, it proved possible to arrive at a common set of design rules and a common set

of test devices and circuits, making joint design and exchange of information possible.
At Philips the first results for BICMOS1 P, have been obtained, the results generally were in fair agreement with the theoretical predictions. A complete characterisation of the process is now being carried out and some circuits, including those for the demonstrator are being designed.

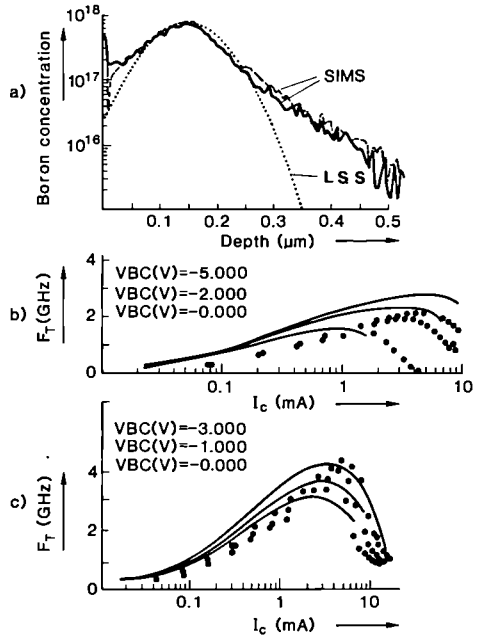


fig. 4 a) Boronconcentration in the base of the BICMOS NPN transistor. Dotted line predicted by LSS and SUPREM. Measurements by SIMS.
b) and c) After fitting the boronconcentration in SUPREM, NPN cutt off frequency F_T of respectively lowly doped (NWS) and normal well (NW) is presented. Drawn curves are measured.

Now, at the end of year two, the first very encouraging results of BICMOS 1 have been obtained at Siemens, and there is good reason to expect that the objectives of BICMOS 1S - viz. a bipolar transit frequency of 3 GHz and a CMOS toggle frequency of 500 MHz - will be attained reliably by the end of year 3.

Meanwhile in the second year a start has been made with BICMOS 2, a process similar to BICMOS 1 but denser and more particularly having enhanced bipolar capability to emitter widths below 1/um. The processes are now moving to be more alike; at both firms a LDD structure, (cf. fig. 2), in the CMOS part will be used and more importantly a second polysilicon layer to realise high performance emitters in the bipolar transistors (cf.fig. 5). Such emitters are already used throughout the world for bipolar only processes.

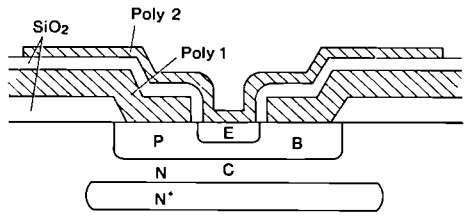


fig. 5 Bipolar transistor using double polysilicon layers as envisaged in BICMOS 2P

Finally it should perhaps be noted that the work referred to in the papers 3, 4, 5, 6, 7 and 8 is at least to some extent a consequence or even a part of the BICMOS project.

References

1) I.Fukushima, K. Kuwahara, K. Hoye, N. Horee, K. Itoigawa, S. Ichinura, M. Nagata. A BIMOS FET Processor for VCR audio, IEEE, ISSCC 1983, p 242-243

2) F. Rausch, H. Lindeman, W.J.M.J. Josquin, D. de Lang and P.J.W. Jochems. An analog BIMOS technology Extended abstracts of the 18th Conference on Solid State Devices and Materials Tokyo 1986, pp 65-68

3) P.A.H. Hart, Philips, K. Bürker and A. Wieder Siemens, BICMOS 412. Proceedings ESPRIT Technical Week Brussels 1986.

4) J. Winnerl and E.P. Jacobs, Proceedings ESSDERC Aachen, 1985.

5) D. de Lang and W.J.M.J. Josquin, Optimization of a 1.5/um BICMOS process. BICMOS Symposion, Abstract no. 275. Electrochemical Soc. Philadelphia. May 1987.

6) J. Winnerl, F. Neppl, B. Vollmer and M. Stegherr. The usefulness of advanced drain structures as emitters in scaled BICMOS, BICMOS Symposion, Abstract no. 606 RNP. Electrochemical Soc. Philadelphia, May 1987.

7) M.F.C. Willemsen, A.E.T. Kuiper, A.H. Reader, R. Hobbe, and J.C. Barbour. In situ investigation of TiN formation on top of TiSi. Submitted to Journ. Appl. Phys. 1987.

8) P.A.H. Hart. BICMOS, Dream or nightmare, Proceedings ESSDERC conference Bologna 1987.

Project No. 281

SUBMICRON BIPOLAR TECHNOLOGY II

A 3 ns access time 4K bit ECL RAM with an optimized cell design

W.M. Werner, K.R. Schön, K. Delker, A. Glasl, K. Wiesinger

SIEMENS AG

Introduction

The project "Submicron Bipolar Technology" started on 1.1.85 and ends on 31.12.89. It focusses on the development of a bipolar technology for high speed data and signal processing products. This technology will allow the realization of gate arrays with a complexity of 25K gates. The gate delays will be <100 ps and the speed power product <0,1 pJ. The feasibility of IC's with the indicated integration level will be demonstrated at the end of the project by an adequate test vehicle (demonstration chips). Previously the state of technology development is demonstrated at appropriate stages by 2 more demonstration chips. The first demonstrator which is a 10K gate array was available 1,5 years after starting the project.

The second demonstrator is a 4K bit ECL RAM with 3ns access time and on chip latches developed for cache memories and control store in computer applications. First silicon of this device was availlable on schedule in March 1987. The dynamic properties of these first samples conform completely to the specification. This paper describes the memory cell, the process technology, the process- and device modeling and the circuit design of the memory.

Memory Cell

The memory cells of bipolar high speed RAM's are emitter coupled for fast read/write operation and Schottky clamped to prevent the transistor from saturation. In general a high ratio $\beta=(2C_{CB}+C_{SBD})/C_S>5$ is necessary to reduce the access time and to improve the cell stability, where C_{CB} is the collector base capacitance, C_{SBD} the Schottky diode capacitance and C_S the collector-substrate capacitance of the transistor (Fig. 1).

A high collector node capacitance $C_C=4C_{CB}+C_{SBD}+C_S+2C_{EB}>800fF$ is also important to assure sufficient alpha particle immunity, where C_{EB} is the emitter-base capacitance of the transistor. Usually these requirements are fulfilled by increasing the SBD capacitance C_{SBD} or by adding a Ta_2O_5 film capacitor in parallel to the SBD /1 - 3/.

In the proposed cell (fig. 1), an external capacitor C was introduced between the collector nodes of the cell transistors to fulfill the mentioned requirements. This capacitor is realized using highly doped polysilicon layers for the electrodes and a 20 nm thick silicon nitride film as dielectric. This concept has several advantages:

a) The collector-substrate capacitance C_S can be reduced because the said capacitor is placed outside the active transistor area.
b) Only half of the additional capacitor value is necessary for the same ratio ß when placing this capacitor between the collector nodes instead of connecting it in parallel to the two SBD´s.
c) The external capacitor has no alpha particle sensitivity.
d) Due to the Miller effect and the special arrangement of the external capacitor only a fourth of the additional capacitance value is necessary for the same alpha particle immunity compared to the concepts mentioned in /1 - 3/.

Process Technology

The technology is based on the OXIS- processes described in /4 - 6/. Lateral scaling down to a minimum feature size of 1,0 µm is achieved by the use of 5 : 1 lithography in conjunction with 1 : 1 projection for less critical layers. For the isolation a conventional LOCOS process including a planarisation step is used. Besides lateral scaling down, transistor performance was improved by a reduction of both, the epi thickness and the base-emitter junction depths. Table 1 shows typical process parameters and table 2 shows device parameters for minimum transistor. Polysilicon layers are used for the electrodes of the external capacitor, the polysilicon emitter and all resistors. For the SBD, PtSi is applied as contact material. The two-layer metallization

uses a polyimide-plasmanitride sandwich with its excellent planarizing nature as the dielectric. This allows the utilization of non-nested vias with respect to the first metal layer /6/.

Modeling

Process, device and circuit simulation was used to develop technology and circuit in parallel. The aims were an optimized bipolar process for the required chip performance and a stable technology founded on a sensitivity analysis of device characteristics with respect to realistic process parameter fluctuations.

Since actual simulation tools in general give only qualitative results, we adapted process-dependent models and parameters to the technology in question. For process modeling the 1-dimensional program SUPREM III /7/ was used. The models for ion implantation at low energies /8/ and diffusion at low temperatures were studied and improved to give good agreement with SIMS measurements of shallow boron and arsenic profiles. Device simulation was done in MEDUSA /9/, where the influence of polysilicon on emitter properties was represented by modifications of recombination effects. With the resulting network simulation parameters the circuit was designed in SPICE II.

Limited by disposable equipment we modeled the desired buried layer, epi, base and emitter doping profiles of the basic transistor using SUPREM III. The final doping concentrations at the end of process were coupled by software link to MEDUSA. The MEDUSA output (charge, current and field distributions) was converted into network simulation parameters to get a better description of the devices for optimization. Tab. 3 as an example shows the influence of epi doping concentration and width on various parameters with a given emitter-base-profile. Base charge and transit time depending on ion implantation and annealing conditions were studied in detail. Estimates on the influence of 2-dimensional effects like the emitter edge situation were realized by using quasi-2-dimensional calculation in MEDUSA. Process definition ended with a first set of SPICE-parameters, which was the basis of the initial circuit design. Comparison to hardware results shows reasonable agreement on all three stages of simulation.

Circuit Design

The memory array consists of 64 x 64 cells organized as 1K x 4 bit. Two-stage word and bit decoder with high-speed circuits ensure minimum access time with moderate power consumption. The RAM is fully compatible with ECL 100k.

To replace off-chip latches needed for synchronous operation /10/ all input and output buffers are provided with latches. The latches are connected in parallel to the amplifiers thus not affecting the access time in conventional RAM mode. The block diagram of the synchronous RAM is shown in fig. 2. In the low state of the clock, all input latches are transparent and the outputs are latched and the reverse condition occurs in the high state of the clock leading to minimal cycle time. An on-chip write-pulse generator supplies the read/write control circuit with a write pulse independently of the external write-pulse width. For operation in conventional RAM mode, the clock input is at constant low state (or may be let open). Also the low state of two control inputs, WEMOD and CLKMOD, is disabling the write-pulse generator and turning the output latches to the open state, respectively.

The RAM has a typical access time of 3.0 ns (see fig. 3) and a minimum write-pulse width less than 2 ns. The minimum cycle time is about the same as the access time, the power dissipation is 1.5 W typically. Table 4 summarizes the main features of the RAM. Fig. 4 shows a micro-photograph of the memory.

REFERENCES

/1/ K. Ogine et al., "Technology improvement for high speed ECL RAMs", IEDM Techn. Dig., p. 468 - 471, Dec. 1986

/2/ H. Miyanaga et al., "A 1,5 ns 1K Bipolar RAM using novel circuit design and SST - 2 technology", IEEE J. Solid-State Circuits, Vol. SC - 19, p. 291 - 298, June 1984

/3/ J. Nolcubo et al., "A 4,5 ns access time 1K x 4 bit ECL RAM", IEEE J. Solid-State Circuits, Vol. SC 18, p. 515 - 520, Oct. 1983

/4/ E. Gonauser et al., "A Bipolar 230 ps Masterslice Cell Array with 2600 Gates", IEEE J. Solid-State Circuits, Vol. SC 19, p. 299 - 305, June 1984

/5/ H. Ullrich et al., "A 150 ps 9000 Gate ECL Masterslice", IEEE J. Solid-State Circuits, Vol. SC 20, p. 1032 - 1035, 1985

/6/ H. Eggers et al., "A Polyimide-Isolated Three-layer Metallization System for Bipolar Gate Arrays", Siemens Forsch. u. Entwickl.-Ber., Bd. 15, Nr. 2, p. 64 - 67, 1986

/7/ C.P. Ho, J.D. Plummer, S.E. Hansen, R.W. Dutton, IEEE Trans. Electron. Devices, ED 30, 143B, 1983

/8/ H. Ryssel, K. Haberger, K. Hoffmann, G. Prinke, R. Dumke, A. Sachs, IEEE Trans. Electron. Devices, ED 27, 1484, 1980

/9/ W.L. Engl, R. Laur, H.K. Dirks, IEEE Trans. on CAD, CAD-1, 85, 1982

/10/ K. Yamaguchi et al., "An ECL 4K bit bipolar RAM with an effective access time of 2.5 ns and on-chip address latches", Sympo. VLSI Tech. Dig. Papers, p. 52, 1984

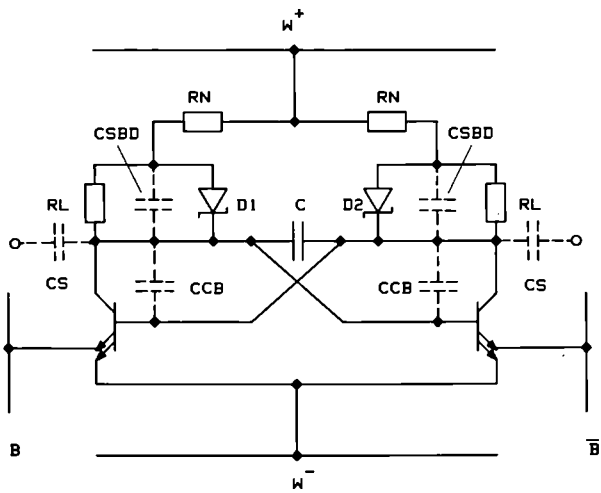


Fig. 1: Memory cell with external capacitor C

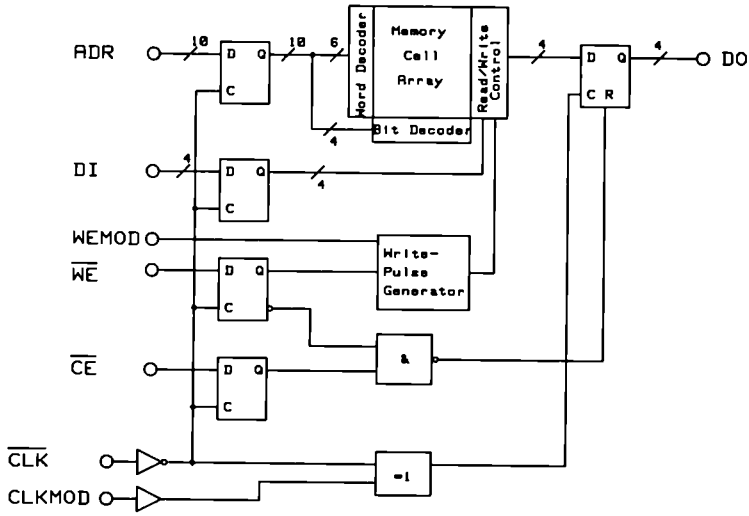


Fig. 2: Block diagram with on-chip latches

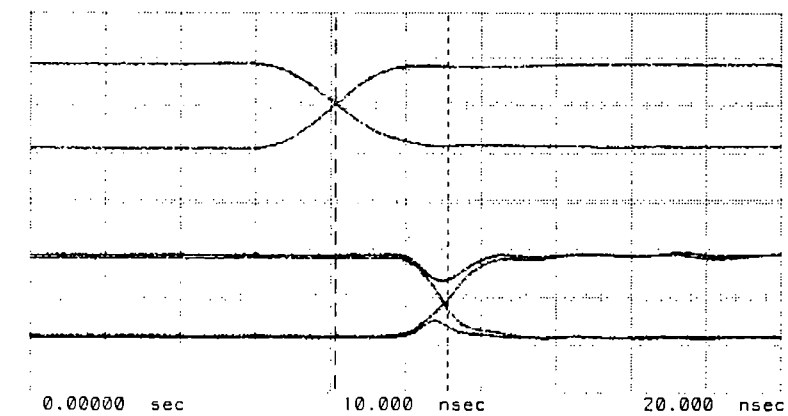


Fig. 3: Address input and data output waveforms of the RAM

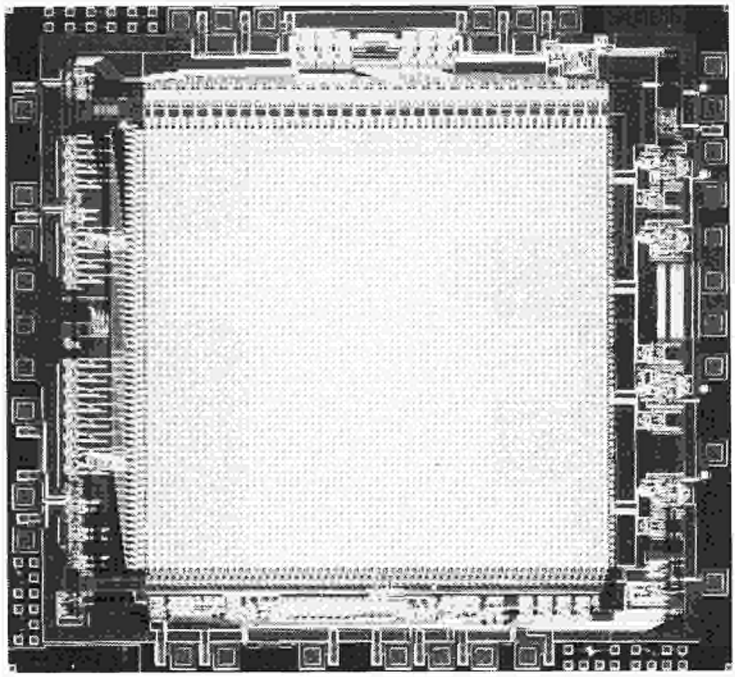


Fig. 4: Microphotograph of the RAM

Isolation	Locos
Minimum feature size	1,0 µm
First metal layer pitch (incl.vias)	5,0 µm
Epi layer thickness	1,0 µm
Emitter depth	0,2 µm
Base depth	0,4 µm
Polysilicon layer 1	0,2 µm
Polysilicon layer 2	0,25 µm

Table 1: Process parameters

Emitter area	4x1 μm^2
Base-emitter capacitance	21 fF
Collector-base capacitance	30 fF
Collector-substrate capacitance	60 fF
Base resistance	490
Cut-off frequency	8 GHz
External capacitor	2,7 $fF/\mu m^2$

Table 2: Device parameters for minimum transistor

ϱ_{EPI} [cm^{-3}]	0.80 E 16	1.60 E 16	3.20 E 16	1.60 E 16	1.60 E 16
d_{EPI} [µm]	1.1	1.1	1.1	1.0	1.2
I_S [A/μ^2]	1.17 E-18	1.25 E-18	1.43 E-18	1.27 E-18	1.26 E-18
J_{BF} [A/μ^2]	1.07 E-3	1.17 E-3	1.31 E-3	1.25 E-3	1.10 E-3
V_{BF} [V]	34.0	29.6	22.4	24.8	32.0
T_F [ps]	13.5	12.5	11.0	12.0	12.5
β	77.9	82.5	91.4	82.8	82.4
C_{JBC} [fF/μ^2]	0.32	0.38	0.50	0.42	0.38

Table 3: Influence of epi doping concentration on device parameters

Organization	1kx4
Input and output level	ECL 100K
Power dissipation	1.5 W typ.
Address access time	3.0 ns typ.
Write pulse width	2.0 ns min.
Chip-select access time	1.6 ns typ.
Cell size	1690 μm^2
Chip size	18.0 mm^2

Table 4: Characteristics of the RAM

Project No. 243

AN ADVANCED BIPOLAR PROCESS USING TRENCH ISOLATION AND POLYSILICON EMITTER FOR HIGH SPEED VLSI

Maurice DEPEY, Pierre SCHOULER, Marcel ROCHE
THOMSON SEMICONDUCTEURS, BP 200 38522 Saint Egrève, France

Peter HUNT
PLESSEY RESEARCH Ltd, Caswell Towcester, Northants NN12 8EQ, UK

Aachim HEFNER
TELEFUNKEN Electronic GmbH, Theresienstrasse 2, D1700 Heilbronn, RFA

1. INTRODUCTION

A new bipolar process has been set up within the ESPRIT project N° 243, "Submicron bipolar technology".
The overall objective of the five-year programme of this project is to develop the capability of manufacturing submicron structures as required for very high speed and very large scale integrated circuits. The effort devoted to this programme is approximately 100 man x year.
In the way towards submicron minimum features, an intermediate stage of developing and demonstrating a one micron process was considered to be necessary. This process, which will be described hereafter, proves to be quite fitted to applications requiring less than 200 ps gate delays and very high packing density, such as signal processors, gate arrays, PROMs, etc.

2. MAIN FEATURES OF THE PROCESS

In addition to high resolution photolithography allowing small sizes and therefore small parasitic capacitances, the following main features contributed to good performances :

- . deep trench isolation
- . polysilicon emitter
- . tri-layer interconnections.

The schematic cross-section of the NPN transistor in figure 1 shows the basic structure (last interconnecting layer omitted).

3. DEEP TRENCH ISOLATION

The lateral isolation between components is provided by deep trenches. The anisotropically etched trenches are 1.2 µm wide and 5 µm deep. They are filled with polysilicon, isolated from the substrate by a thin oxide layer. Since they are deeper than the buried layer, a full wafer buried layer could be adopted, thus avoiding photolithography and etching step prior to epitaxy, for the benefit of yield and simplicity.

One of the problems that had to be overcome is the generation of defects due to stress during the trench filling and more specifically during later oxidation step. A solution was found which allows walled emitter structures thus increasing the packing density.

To complete the isolation, a channel stopper is necessary. Its localisation at the bottom of the trench is another problem : a parasitic P doping on the

vertical edges around the trench would add a parasitic collector-base capacitance to the walled-base NPN transistor. From the measured value of this parameter as compared to the theoretical value, it may be inferred that this problem has also been satisfactorily solved.

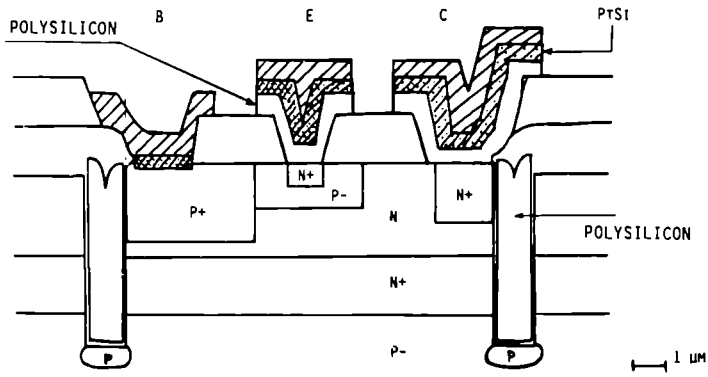


Figure 1 - CROSS SECTION OF THE NPN TRANSISTOR IN THE 1 µm ESPRIT TECHNOLOGY (MASK SIZES)

4. POLYSILICON IN THE EMITTER

Adding a polysilicon buffering layer between the metallic emitter contact and the emitter-base junction is known as a good solution for the difficult trade-off between cut off frequency, current gain and pinched base resistance. A simple reduction of the emitter-base junction depth as required to reduce the parasitic stored charge of minority carriers in the emitter region and therefore increase the speed performance, would have a detrimental effect on the emitter-base injection ratio and hence on the current gain.

The additional polysilicon layer in the emitter introduces some degree of freedom in the compromise between charge and current of minority carriers in the emitter region, because of the poly/mono interface barrier. Many studies have been carried out on this subject in several semiconductors laboratories all around the world in recent years showing that, actually, several effects and interpretations are possible.

We also studied different solutions to reach this composite emitter-base structure. The investigated parameters were : morphology of deposited silicon, surface preparation prior to this deposition, ion implantation and annealing conditions. We finally optimised a process where the polysilicon layer has a buffering effect during As+ ion implantation and allows, via subsequent small diffusion, a very shallow monosilicon emitter zone.

Two other benefits of polysilicon were used :
- its capabilities for connections, enhanced by polycide (PtSi) formation
- its capabilities for resistors implementation.

5. INTERCONNECT SYSTEM

The interconnect system employs one polycide level and two metal levels. The pitches are 2.5 µm, 4 µm and 6 µm respectively. Plasma CVD SiO2 isolates metal 1 from polycide. A planarising layer formed by a sandwich of polyimide and SiN insures the isolation between metals 1 and 2.

6. RESULTS

A first test mask has been used to process wafers. It includes many basic structures and components with size variations, and also ring oscillators and frequency dividers.
A SEM cross-sectional view of a NPN transistor is shown in figure 2.

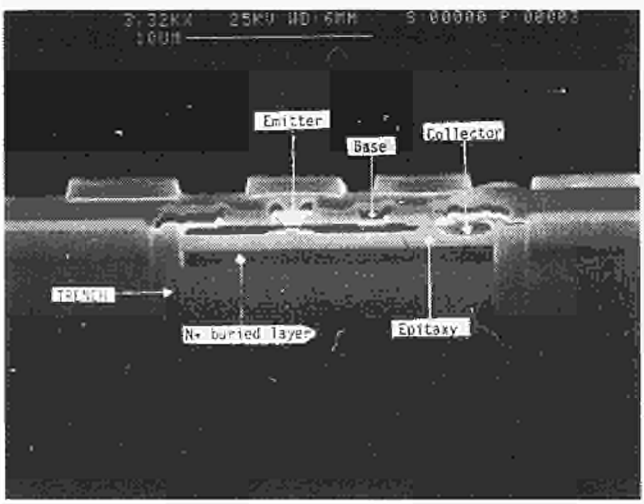


Fig 2 - SEM CROSS-SECTIONAL VIEW ON NPN TRANSISTOR

A tentative transistor referenced T6 with emitter width (0.75 µm drawn size) smaller than the preliminary design rules proved to operate satisfactorily as shown by the curves of figures 3 and 4.

Table 1 shows the measured capacitances for this transistor and for larger ones, as used in the ring oscillators. Also given in this table is the size of the junctions for each transistor.

The cut off frequency Ft (measured on transistor large enough to eliminate some measuring parasitic effects) is 11 GHz. On 21-stage ring oscillators, made of differential ECLinverters with 400 mV nominal differential logic swing, the measured propagation delay time and speed-power product are 100 ps and 0.2 pj respectively. Table 2 shows the effect of transistor geometry on delay times.

TABLE 1 - JUNCTION SIZES AND CAPACITANCES

Transistor ref		T6	TRT 2	TRT 3
BE	area capacitance	0.75 x 1.5 = 1.125 µm² 4.2 fF	1 x 2.5 = 2.5 µm² 16.2 fF	1 x 5.5 = 5.5 µm² 28 fF
BC	area capacitance	7 x 1.5 = 10.5 µm² 7.1 fF	11.75 x 2.5 = 29.4 µm² 19.5 fF	13 x 5.5 = 71.5 µm² 25 fF
CS	area capacitance	11 x 1.5 = 16.5 µm² 17 fF	15.75 x 2.5 = 39.4 µm² 25 fF	16.25 x 5.5 = 89.4 µm² 32 fF

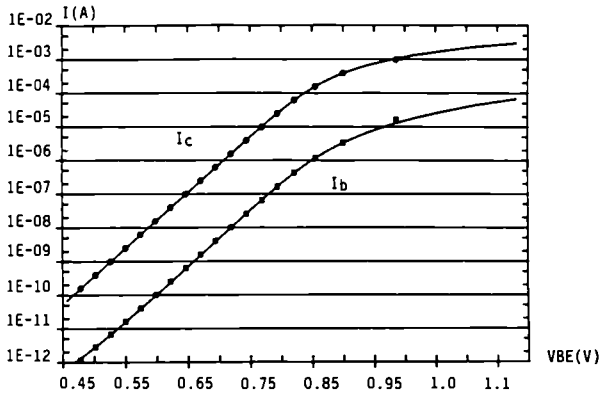


Figure 3 - "GUMMEL PLOT" OF THE SMALLEST NPN TRANSISTOR
(T6 - 0.75 x 1.5 µm emitter)

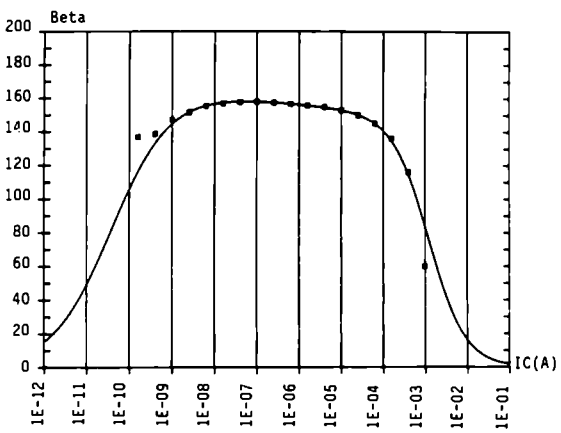


Figure 4 - CURRENT GAIN BETA VERSUS COLLECTOR CURRENT FOR THE
SMALLEST TRANSISTOR (T6 - 0.75 x 1.5 µm emitter)

7. FUTURE WORK

A second test mask has been designed to demonstrate the capability of this technology in terms of yield on very complex circuits. This demonstrator is a long shift register (more than 2 000 stages) made with identical small cells and corresponding to an equivalent complexity of about 17 000 gates.
The maximum clock frequency expected is higher than 700 MHz.

Further improvements of the transistor characteristics are now investigated through novel emitter-base structures. Self-aligned base contact and emitter, with polysilicon access to both emitter and base regions are being experimented. Our target for 1990 is a capability of 50 ps propagation delay time with at least a 20 000 gate complexity, and at a reasonable power consumption.

Table 2 - PROPAGATION DELAY TIME IN THE RING OSCILLATORS FOR DIFFERENT TRANSISTOR GEOMETRIES

Ring oscillator type	Transistor			Emitter size (µm)	Propagation delay/stage (ps)	Current/ stage (mA)
	Ref	Geometry				
1	TRT1	WE	-	1 x 5.5	110	1
2	TRT2	WE	-	1 x 2.5	120	0.4
3	TRT3	WE	LOCB	1 x 5.5	105	1
4	TRT4	-	LOCB	1 x 5.5	100	1

(WE = walled emitter, LOCB = walling locos oxide between collector contact and base region)

ACKNOWLEDGEMENT

The authors would like to thank G. BOREL, D. CELI, JM CHATEAU, JL IMBERT, X. MARTEL, B. SOLIGNAC, D. THOMAS, L. FRITSCH and their associates for the contribution to these results.

Project No. 843

A GaAs 4-STAGE SERIAL MULTIPLIER IN SELF-ALIGNED TECHNOLOGY

M.J. Agnew, J.Puleston Jones, S.W. Bland

STC Technology Ltd, London Road,
Harlow, Essex. CM17 9NA, England

1. INTRODUCTION

A cascadable 4-stage serial multiplier has been designed and implemented using advanced VLSI semi-custom design techniques based on a library of standard cells. The logic circuitry (414 standard logic gates) was automatically layed-out in full. The circuit has been fabricated using 1 μm gate length SAINT DCFL MESFETs. The input and outputs are ECL100K compatible. The input data can be in 2's compliment or magnitude only format. The output product can be either truncated or rounded. The circuit operates on-wafer up to 800 MHz. Ring-oscillators on-wafer operate with a propagation delay of 128 ps (fanin=1, fanout=2) and a power dissipation of 0.59 mW/gate.

2. CIRCUIT DESIGN

The circuit for each stage of the multiplier (Fig. 1) uses a shift and add algorithm[1,2].

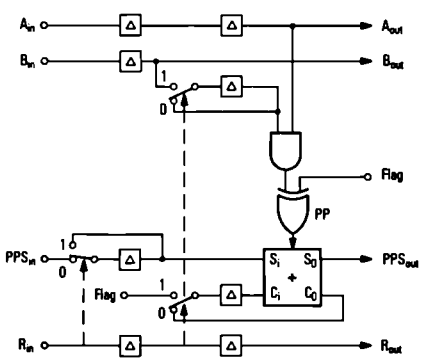


FIGURE 1
Serial Multiplier Circuit per Stage

The input data is supplied to each stage in serial form from the previous stage. Each bit of the input multiplicand word (B) is latched in subsequent stages of an n-stage multiplier. The input data word (A) can be any number of bits, with the restriction that MSB must be repeated in order to avoid internal overflow. The partial product (PP) formed by each stage is evaluated by multiplying (logic AND) each bit of the input data word (A) by the latched bit of the multiplicand (B). The partial product (PP) is combined with the partial product sum of the previous stage (PPSin), using a full ADD with CARRY block to form the partial product sum of a stage (PPSout). The shift-left one bit in significance for subsequent partial products (LSB to MSB) of normal multiplication, is overcome by shift-right one bit and truncation of the previous partial product sum (PPSin) with respect to the partial product (PP). Each data word is distinguished by supplying a high (TRUE) on the data latch line (R) in synchronisation with the LSB of the word. For magnitude only input data format the flag line is held low (FALSE). The negative weighting of the MSB partial product in 2's compliment input data format is achieved by holding the flag line high (TRUE) on the final multiplier stage.

The circuit was implemented using semi-custom in-house design techniques originally developed for Si CMOS circuits (Fig. 2).

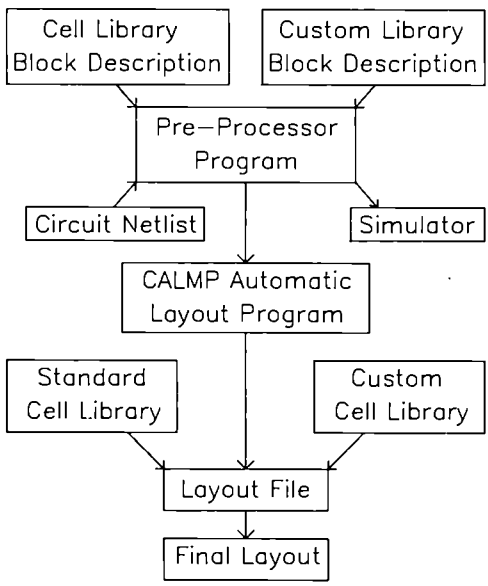


FIGURE 2
STC Automatic Cell-Based Layout

The pre-processor program filters the data so that only the data required by a tool is supplied to it in the format that it expects. The circuit netlist describes the circuit in terms of standard, and circuit specific cells, described in the block description files. Once the circuit netlist is generated, the simulator allows the performance of the circuit to be verified. Once this is achieved, the autolayout software is used to create the layout of the logic circuitry, subject to predefined criteria. Additional features are manually added to complete the circuit layout, these include: power lines, output buffers and bond-pads. In addition to the 4-stage multiplier, a 1-stage multiplier, 15-stage ring-oscillators and process test structures were included in the final layout.

The process employed to fabricate the circuits used a self-aligned gate technique known as SAINT originally developed by NTT in Japan[3]. A photograph of the fabricated 4-stage multiplier circuit is shown in Fig. 3.

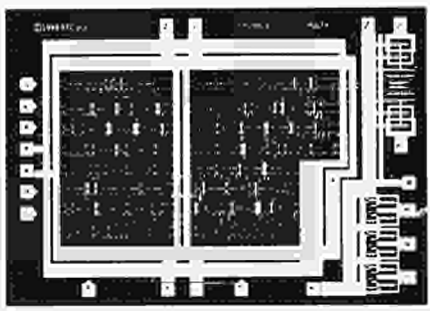


FIGURE 3
4-Stage Serial Multiplier

3. TEST RESULTS

All circuit testing to date has been performed on-wafer. DC FET device threshold voltages were measured as +0.243±0.048 V for EFETs, and -0.894±0.088 V for DFETs. Ring-oscillators with fanout=1, and a fanin of 1 and 2 showed a propagation delay of 86±9 ps/gate. Whereas those with a fanin=1, and fanout=2, had a propagation delay of 128±12 ps/gate.

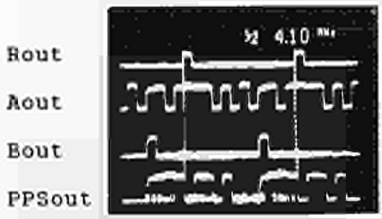


FIGURE 4a
1st Stage Fully Functioning at a Clock Freq. of 50MHz:

FIGURE 4b
Partial Function at a Clock Freq. of 800 MHz: AIN=10 Flag=0 PPSin=101011000000

All functions of the 4-stage multiplier have been verified at a clock frequency of 50 MHz with the aid of a HP8018 word generator (Fig. 4a). All stages of the multiplier functioning can be displayed up to a clock frequency of 320 MHz, using a 8-bit word generator. Above this limit only partial circuit functional operation can be displayed using combinations of pulse generators. This partial function has been displayed up to a clock frequency of 800 MHz (Fig. 4b), using 6 dBm of clock signal to overcome the -10 dB signal insertion loss of the probecard above 600 MHz. The power dissipation of the logic circuit was 670 µW/gate at ECL100K levels.

4. CONCLUSION

A GaAs 4-stage serial multiplier has been designed using DCFL, implemented using semi-custom standard cell library techniques, and successfully fabricated with 1 µm gate SAINT. Full functional operation has been displayed up to a data rate of 320 Mbit/s, partial operation up to a data rate of 800 Mbit/s has been verified. The logic circuitry has a power dissipation of 300 mW at ECL100K I/O levels. The logic circuitry has 414 standard gates all automatically layed-out in full. The ring-oscillator results, e.g. 128 ps (fanin=1, fanout=2), combined with circuit analysis suggest a circuit data transfer rate in the region of 1 Gbit/s is possible. The partial functional operation at 800 Mbit/s agrees well with these predictions.

5. ACKNOWLEDGEMENT

This work was partly funded by the EEC under ESPRIT project 843.

6. REFERENCES

[1] Myers, D.J., 'Multipliers for LSI and VLSI signal processing applications', MSc Report MSP5, University of Edinburgh.

[2] Lyon, R.F., 'Two's compliment pipelined multipliers', IEEE Trans. on Commun. Sci. , pp418-425, April 1976

[3] Yamasaki, K. et al., 'Self-align implantation for n^+ layer technology (SAINT) for high-speed GaAs ICs', Electronics Lett., 18, No. 3, pp119-21, February 1982.

Project No. 843

THE DEVELOPMENT OF A TUNGSTEN SELF-ALIGNED GATE PROCESS FOR GaAs MESFETs

I. Davies, K. Vanner, J. Cockrill, B. McAllister

Plessey Research Caswell Ltd., Allen Clark Research Centre, Caswell, Towcester, Northamptonshire, NN12 8EQ.

This paper will describe the development of a tungsten gate metal technology for the production of self aligned GaAs FETs (SAGFETs). The realisation of tungsten films with low stress and low resistivity have enabled the production of both enhancement and depletion mode transistors. Good diode characteristics are shown by these films with barrier of heights of 0.85V and ideality factors of 1.1.

Three main topics were studied in developing the technology, namely metal deposition, metal etching and the annealing properties of the films. Both the deposition and etching processes are capable of damaging the active layer and, therefore, particular attention was given to minimising ion bombardment damage.

Several methods for the deposition of the tungsten metal were investigated, these were PECVD, e-beam evaporation, d.c. and r.f. sputtering. PECVD tungsten films formed from the decomposition of WF_6/H_2 mixtures were evaluated. It was found that the adhesion of these films to GaAs was poor. The resistivity of the deposited films were measured as a function of anneal temperature and a dramatic change was observed at 730°C. The resistivity value of 100μΩ cm falls to a value of 15μΩ cm indicating that a phase change may have occurred. The diode characteristics show no change upon annealing, with values of ideality factor 1.40 and barrier heights of 0.56V being measured for both the as deposited and annealed devices.

There were problems encountered with e-beam evaporating tungsten films owing to the lack of freedom in varying the process parameters in order to tailor the properties of the deposited layer. Although low resistivity values were obtained by this method, the films were found to be highly stressed. Improved barrier heights were seen with these films, of the order 0.7V, however, the ideality factor was still poor.

Both d.c. and r.f. sputtering of tungsten was assessed and this deposition method was found to give greater flexibility in the control of the resistivity and stress of the film produced. Low resistivity films could be obtained with d.c. sputtering, however, the adhesion of the film to the GaAs surface was found to be a major problem. The physical properties of r.f. sputter deposited films were studied as a function of the process parameters; power, pressure and bias. It was found that although pressure and bias conditions had a small effect on the film produced, the power had a very much greater influence on the resultant film properties. A low power process gave low stress but high resistivity, while a high power process gave the opposite result. Therefore, in optimising the process a compromise was reached where films with small compressive stress and resistivities of 10μΩ cm (bulk value for W 5.5μΩ cm) were produced.

An important property of the metal contact is its stability when subjected to subsequent n^+ anneal conditions. The diode characteristics were monitored as a function of anneal temperature (Fig. 1). The barrier height shows a gradual

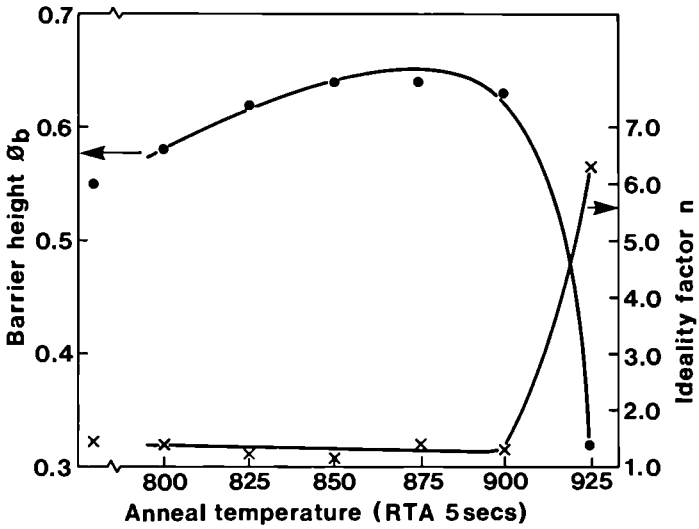


FIGURE 1
Effect of n^+ anneal on diode performance

increase to a maximum value at ~875°C then decreases, whilst the ideality factor remains constant to 900°C then degrades rapidly. The optimum anneal temperature to activate the n^+ implant has been shown to be in the range 875°C-900°C [1], therefore, no compromise in the performance of the n^+ region is required with this contact metallisation. Using this process, planar diodes with barrier heights of 0.85V have been made with ideality factors of 1.1 after annealing. These results are comparable to those reported for similar geometry devices using other refractory gate materials.

Analysis of the films by X-ray diffraction, transmission electron microscopy, and Rutherford back scattering spectroscopy was carried out in order to obtain data on the nature of the tungsten-GaAs interface, and to assess how it is affected by sputtering conditions.

X-ray diffraction studies showed a preferred orientation for film growth of [110] perpendicular to the substrate surface. No lattice changes occur during anneal cycles up to 900°C. Also similar crystal structure is seen for high and lower power deposition. The TEM results indicated that the low power deposition gave a finer grain structure than that seen with the higher powers. No evidence of alloying of the metal film with the GaAs was seen after annealing. The high power films showed evidence of cracks in the film after heating, indicating a high degree of stress present. RBS analysis showed the bulk interface roughening to be <5Å and that large scale penetration of tungsten into GaAs does not occur. An impurity was shown to be present in the lower power deposited films, thus accounting for the high resistivities seen with these films.

The patterning of the tungsten film was another major aspect of the development of the process. The SAGFET technology requires both anisotropic and controlled isotropic etching of the gate structure. A variety of etch gas compositions have been evaluated in the reactive ion etching mode, these include NF_3, CF_4 and SF_6 together with additions of either O_2, H_2 or N_2 in various ratios. The

mask procedure for the etch process was studied with the evaluation of float off and subtractive processes using photoresist, dielectrics and metal mask materials.

The dielectric materials gave poor gate definition after etching as the etch gases tend also to etch the mask material. Both photoresist layers and metal films have been successfully used to fabricate devices. SIMS and Auger evidence suggest however, that metal migration may occur through the underlying tungsten layer and this could possibly have a deleterious effect on device performance.

The process parameters of gas flow, power and pressure were adjusted to give minimum bias voltages (<50V) and hence minimise any induced damage. Under these restrictions, SF_6 was found to give the optimum anisotropic etching, producing near vertical sidewalls with little undercut of the gate stripe. The addition of other gases (O_2, H_2 or N_2) gave an isotropic nature to the etching. The use of oxygen gave a self limiting undercut etch of ~0.1μm, however, mask erosion was a problem. The inclusion of hydrogen gave larger undercut structures, unfortunately poor edge control was seen. Nitrogen gives a very well controlled isotropic etching regime, with a linear undercut etch rate. Good, well defined submicron gate structures can readily be achieved using this process (Fig. 2).

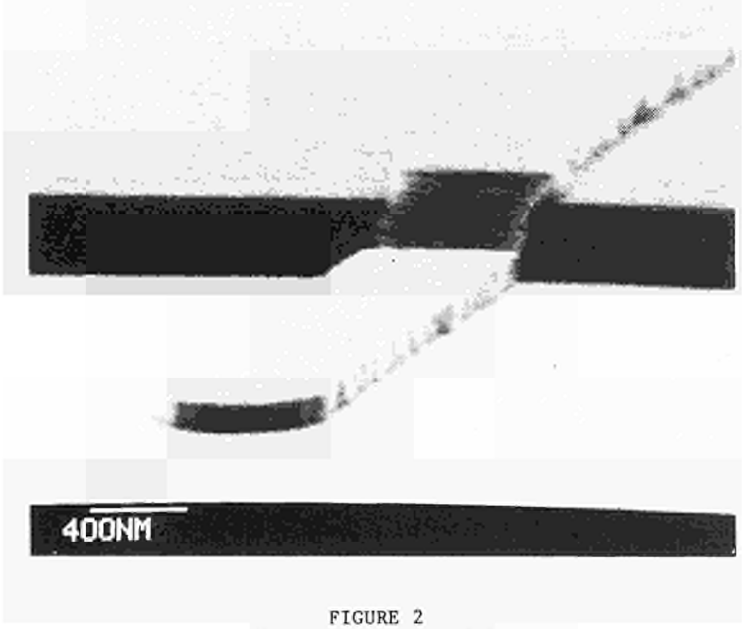


FIGURE 2
Tungsten gate profile

The process technology described above has been used to fabricate both enhancement and depletion mode SAGFETs with sub-micron gate lengths exhibiting good uniformity and high yields. The diode characteristics were good for both types

of device i.e. depletion mode $\emptyset$ = 0.67V n = 1.14, enhancement mode $\emptyset$ = 0.72V n = 1.13. The FET characteristics were good with a transconductance of 170mS/mm seen for the depletion mode devices with a greater than 70% yield (Figs. 3a and 3b). The enhancement mode material ($V_{th} \approx 0.25V$) is a particularly good monitor of process induced damage as the active layer is very thin. This result, therefore, emphasises the fact that a low damage process has been successfully developed.

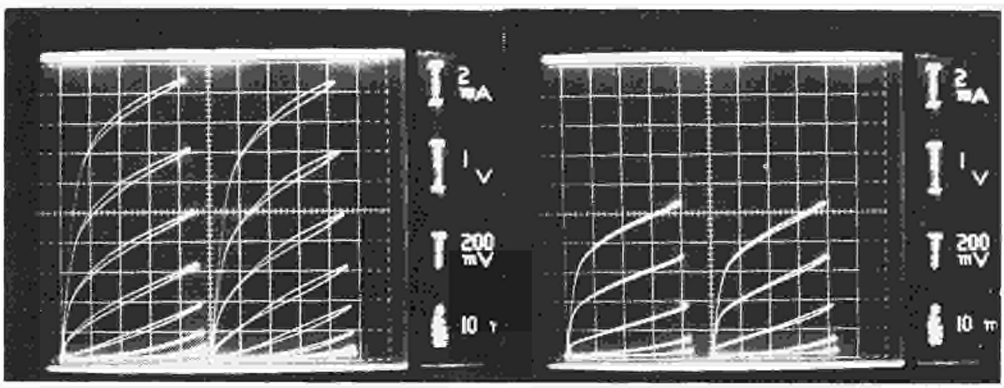


FIGURE 3a
D-FET characteristics (light, dark)

FIGURE 3b
E-FET characteristics (light, dark)

In conclusion the above development has produced a high yield process utilising the advantages of a pure refractory gate metal. The use of pure tungsten exhibits the advantage of low resistivity and high temperature stability coupled with a less complex technology when compared to the use of composite target or co-sputtering used by other laboratories to produce refractory type contacts. It has been shown that r.f. sputter deposition gives the optimum in terms of adhesion to GaAs, low stress, low resistivity and good Schottky behaviour. The tungsten GaAs interface shows good integrity of the layers after the n^+ annealing process. A low damage process has been developed and devices with submicron gate geometries have been successfully fabricated. The depletion mode devices gave transconductances of 170mS/mm with a greater than 70% yield.

REFERENCES

[1] Blunt R.T., Annealing Data of Implanted Layers. Plessey Research Caswell Ltd.

ACKNOWLEDGEMENTS

This work was supported under ESPRIT programme 843. Our thanks also go to the Plessey Company for their support, as well as colleagues at STL Technology, Siemens and LEP for their useful discussions and interaction.

Project No. 554

TOWARDS THE 0.7 MICRON SPECTRE CMOS : A 1 MICRON DOUBLE METAL PROCESS

D. BOIS *

CNET/APF, Chemin du vieux Chêne 38243 MEYLAN (F), Project nb 554

1. INTRODUCTION

The goal of the so called SPECTRE project (nb 554) is to carry out, both at MHS and SGS, an industrial demonstration of a 0.7 µm CMOS process by the end of year 1989. In that intent, several basic techniques and process modules were investigated from the beginning of the project i.e. from 1985 by all the partners involved : SGS, MHS, B.T., Bull, IMEC, UCL and CNET. At the end of 1987 an intermediary demo using 1 micron lithography was planned in order to assess some techniques and to demonstrate the capability of the partners to assemble a submicron process within the last two years of the program. This demonstration was expected to be carried out : (i) in the pilot line of CNET for parametric analysis of the various basic techniques, and for processing of building blocks ; (ii) at SGS and MHS to assess other basic techniques and to show the capability of the industrial partners to manufacture complex devices.

It is the purpose of this paper to describe and discuss a few experimental results obtained during the assembling phase of the 1 µm process. A brief description of the 1 µm process used at CNET to test the various process modules, and to demonstrate the capability of our pilot line, will be given first. Then, we shall focus on three major concerns for submicron technology: the device isolation, the optimisation of pMOS transistors, the interconnects scheme. Finally, the demonstration which is going to be completed before the end of 1987 will be described.

1. 1 micron CMOS process at CNET

The features of the proces now in use at CNET are summarized in table 1

Table 1 Key features of CMOS 1 WA1

Substrate	epi p/p^+
Well	n
Isolation	LOCOS
Gate	polycide WSi2
LDD structure	spacers
1st dielectric	BPSG
Reflow and anneal	RTA
1st metal	tungsten
2nd dielectric	planarized PECVD
2nd metal	aluminium

Table 2 and 3 give a brief summary of the design and electrical rules currently used to design the building blocks which will be processed as a demonstrator circuit for SPECTRE.

Table 2 Summary of design rules of 1 micron process

	width µm	pitch µm
Active area	1.4	3.2
n^+ - p^+ distance	4.8	
Gate	1	2.4
Minimum contacts	1	
Metal 1	1.3	3
Vias	1.7	
Metal 2	2.3	4.4

Table 3 Electrical rules

V_{TN} (minimum size)		0.7 V
V_{TP}		0.85 V
Electrical length	NMOST	0.9 µm
	PMOST	1.1 µm
Tox		25 nm
n^+ junction resistance		30 Ω/sq
p^+ junction resistance		50 Ω/sq
Contact resistance to n+		15 Ω
Contact resistance to p+		25 Ω
Gate sheet resistance		4 Ω/sq
1st metal sheet resistance		0.4 Ω/sq
2nd metal sheet resistance		0.4 Ω/sq

The complete set of design rules were discussed and agreed between all partners ; so they take into account the experiences of several partners. They do not always stand for absolute minimum values of the process. The n^+ - p^+ distance is rather conservative because not enough experimental data were available when it was fixed ; this rule will be decreased to 4 µm in the future.

2. DEVICE ISOLATION

The main concerns for submicron devices isolation are : breakdown voltage of the field regions, leakage currents both between Source and Drain of a transistor or between transistors, transistor channel width variation and related narrow channel effect. At the beginning of this project it was decided that trench or box techniques which are studied for 0.7 µm process could not be ready for 1 µm demo. In addition, trenches must be used in conjonction with some kind of LOCOS in order to insure electrical isolation within transistors ; so SUPERPLANOX, which is an SGS proprietary process module, was choosen as the intermediary 1 µm isolation technique.

SUPERPLANOX is basically a LOCOS technique using a thick nitride barrier which reduces the bird's beak to less than 0.3 µm for 0.7 µm thick oxide (as grown at 1000°C). It has been demonstrated by SGS that to take benefit of this short bird's beak, a twin tub process must be used. That is because for single n tub technique the lateral boron diffusion extension is larger than the bird's beak, and therefore, reduces the channel width and leads to strong narrow channel effect.

As twin tub cannot be used in the CNET process, our 1 µm demo will make use of an optimised LOCOS. Actually, figure 1 shows that the effective channel width variations of LOCOS and SUPERPLANOX are identical ($\Delta W \simeq 1$ µm) except when twin is used in conjonction with SUPERPLANOX ($\Delta W \simeq 0.65$ µm). The total channel width variation ΔW can be expressed as follow :
$\Delta W = 2\ Lbb + \Delta W$ (photo) $+ 2\ Ld$ in which Lbb stands for the bird's beak, ΔW (photo) for the dimensionnal variation due to photo-etch, and Ld for the lateral boron diffusion. For the CNET LOCOS process the experimental values for those parameters are : Lbb = 0.35 µm, ΔW photo = 0,2 µm Ld = 0,10 µm.

The dispersion on the overall ΔW is of prime importance, since the bias can be account for by a proper sizing at the mask level ; we measured this dispersion to be : ± 0.08 µm (at one sigma).

Another significant parameter of the lateral isolation is the magnitude of the narrow channel effect ; it is shown in figure 2. It turns out that twin-well plus SUPERPLANOX gives much better results since it leads to threshold voltages constant down to less than 2 µm design rules for the active area of nMOS Transistors. Therefore this last technique will be used for the 1 µm demonstration at SGS. LOCOS will be used for the single well CNET's process ; it will serve as a basis for assessement of other processes which are currently being studied : SILO, trench, minitrench or BOX. First data dealing with SILO assembled at CNET show that this technique can be used to reduce the bird's beak to about 0.2 µm without any defects being generated in the thin oxide.

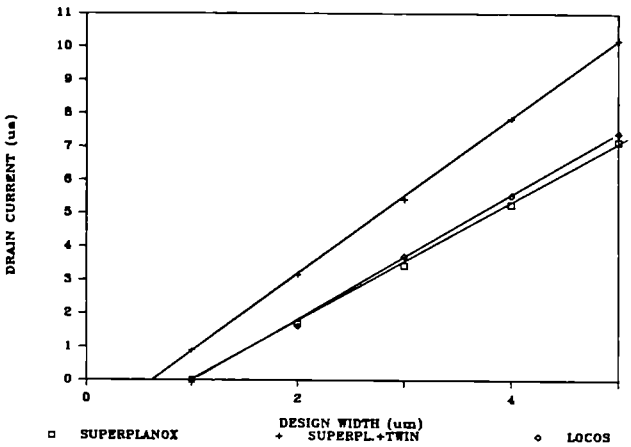


Fig 1 :

Drain current of nMOS transistors vs design width. The intercepts of the curves with the X axis give the value of the difference ΔW between design and electrical width of the transistors.

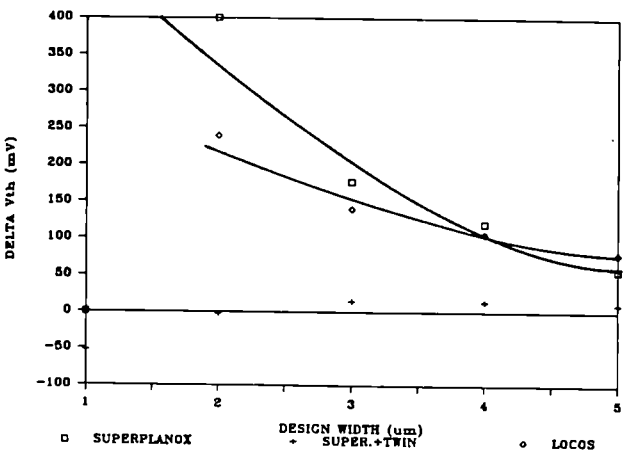


Fig 2 :

Narrow channel effect of nMOS transistors for three isolation and well strategies as described in the text.

At last, isolation between transistors has been characterized for different n+ - p+ distances as shown in figure 3. The breakdown voltages for n and p field oxide parasitic transistors is quite satisfactory even for less than 1 µm design rules.

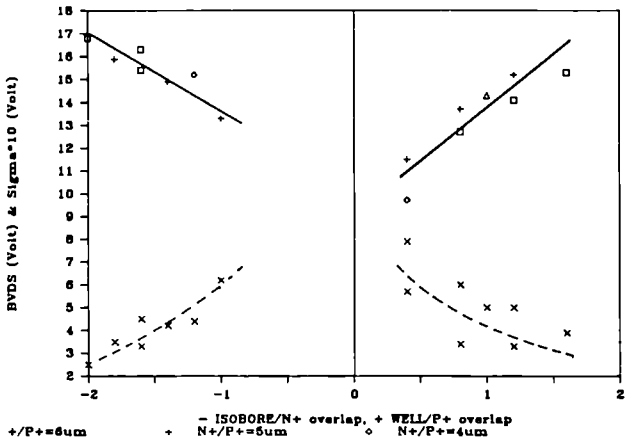


Fig 3 :

Breakdown voltage (BVDS) of field parasitic transistors for different design rules (in µm) : overlap of the isobore around n^+ on the left, overlap of the well around the n^+ area on the right. Different n^+ - p^+ distance are shown by different symbols. The two dashed curres (X) indicate the dispersion (σ x 10 inV) on the BVDS data within a lot of wafers.

3. PMOS TRANSISTOR CHARACTERISTICS

The optimisation of submicron pMOS transistor is difficult since it is basically a depletion device for threshold voltages lower than 1 Volt with n^+ polysilicon gate. So, we investigated retrograde well as compared to conventionnal ones in order to assess their effect on the pMOST.

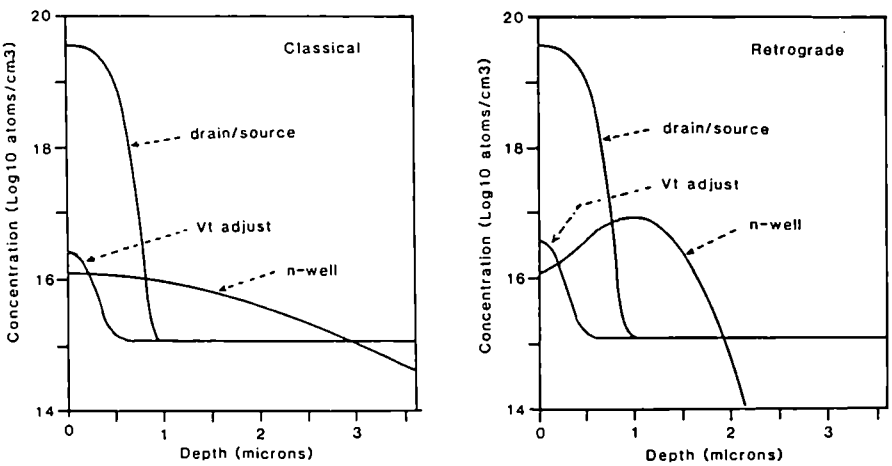


Fig 4 :

Concentration profiles of : boron doped drain/source, threshold voltage implant, and phosphorus doped n well as calculated using TITAN. (after Ternisien et al. ESSDERC 1987).

Figure 4 describes the dopant characteristics used in our experiments. The retrograde n well was obtained by high energy phosphorus implants. The subthreshold characteristics of 50 µm wide pMOS transistors realized in those wells are compared in figure 5.

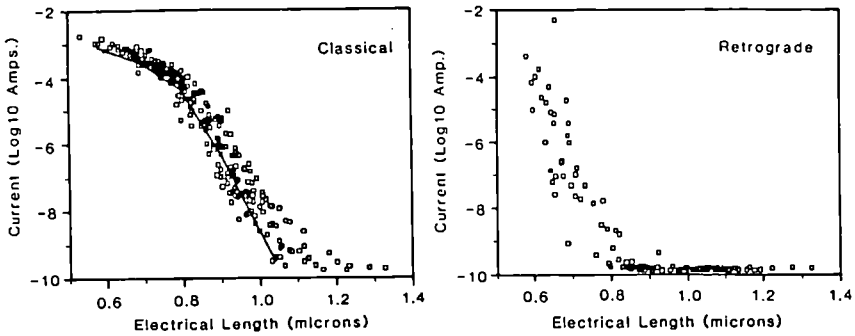


Fig 5 :

Substhreshold current of pMOS transistors (VDS = - 10V, V_G = 0V) for conventionnal and retrograde well. The solid line represents the results from 2D simulator JUPIN. (after Ternisien et al. ESSDERC 1987).

Indeed the retrograde well technology gives better results. For a given acceptable leakage current limit, the gain on the electrical length for pMOS transistors turns out to be : 1 µm for conventionnal well and 0.75 µm for retrograde well. So, such retrograde well could be satisfactory for 0.7 µm CMOS. Anyway, for the 1 µm demo, a conventionnal well is sufficient.

A second approach to submicron devices is the use of refractory gate materials. Tungsten was choosen for that purpose by SPECTRE partners. It will be demonstrated by the end of the year at MHS on actual integrated circuits.

So, at the end of this first phase it will be possible to compare experimental data of four approaches : conventionnal wells at CNET twins-well at SGS, refractory gate at MHS and retrograde well at CNET. This should provide a very good basis to choose which one is the best for 0.7 µm technology.

In addition to this preparation of the future, the work carried out in this first phase of SPECTRE program actually allows us to fabricate devices with the design rules described in tables 2 and 3. Let us notice that those design rules apply for general purpose digital circuits or SRAM. More aggressive electrical lengthes could be used in some cases : e.g. 0.8 µm pMOST electrical length if one tolerates a subthreshold current of 20 nA/µm.

4. INTERCONNECTS

Two interconnects schemes are investigated to prepare submicron SPECTRE process : (i) the conventionnal double Aluminium with emphasis on reliability ; and (ii) a Tungsten/Aluminium system.

After basic techniques analysis by the different SPECTRE tasks, it has been decided to demonstrate the double Aluminium mainly at SGS and the more exploratory W/Al scheme in the pilot line of CNET.

Table 4 summarized the key features of Tungsten as compared to Aluminium when used as a first interconnect level.

Table 4 : Key features of tungsten interconnects

	Sputtered W	CVD W	Al
Contact spiking	no	no	barrier needed
Hillocks	no	no	yes
Electromigration	no	no	yes
Step coverage	medium	good	medium
Planarization	no	yes	no
Selective deposition	no	yes	no
Reflectivity	low	low	high
Grains	small	large	large
Etching	easy	easy	difficult
Resistivity (μohm.cm)	10-17	7-10	3-7

The major drawback of W is, indeed, its higher intrinsic resistivity. However, a more realistic comparison must take into account the increase of the resistance of actual Aluminium based interconnects, because this material can no longer be used alone in submicron devices. One has to add : (i) a barrier on top of contact, (ii) some species such as titatium or copper in order to avoid hillocks formation and electromigration and (iii) sometime a top layer to improve the photo properties of Aluminium. Moreover, one has to decrease the overall thickness of the interconnect when decreasing its pitch: 0.5 μm seems to be a good compromise for 0.7 μm design rules. This results in a significant increase of the average resistivity of Aluminium based interconnects : values up to 7 μohm.cm have been reported. They are indeed, quite close to the one of Tungsten deposited by Vapor Phase (CVD).

The very good photo-etching properties of sputtered tungsten are illustrated by figure 6 : line width control around ± 0.05 μm is obtained down to 2.2 μm pitch (i.e. well below the goal fixed by all partners for the 1 μm demo) even with monolayer resist, and etching with slope control which is necessary in order to make the subsequent dielectric deposition easier.

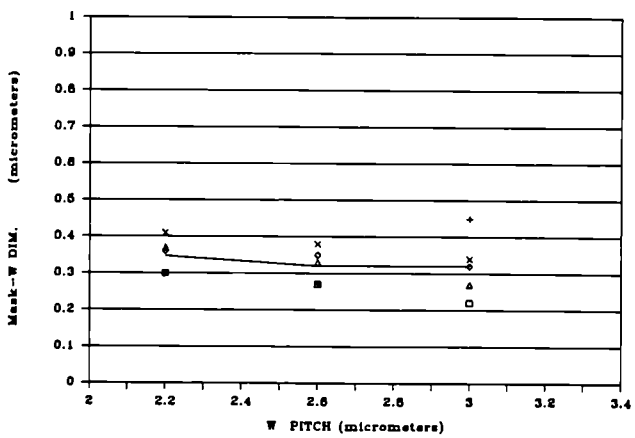


Fig 6 :

Linewidth variation of tungsten interconnects after photo using a monolayer resist and etching with slope control. Each point stand for measurements on a different lot of wafers.

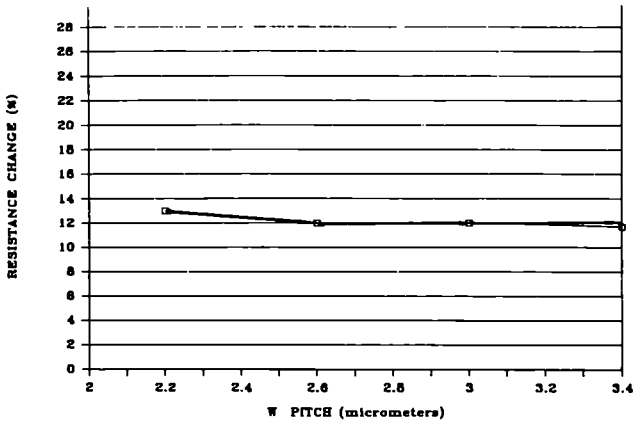


Fig : 7 :

Increase of the resistance of a snake due to non 100 % step coverage for tungsten deposited on top of a grid of polycide covered with BPSG.

The step coverage of the tungsten deposited on top of reflowed BPSG is demonstrated by figure 7. Indeed, the resistance increase of a snake-shaped test structure patterned on polycide/LOCOS steps as compared to the same snake patterned on a flat surface corresponds to both the lines shrinking and thinning at steps. The small variation experimentally measured indicates that both effects are quite small for tungsten.

In order to decrease the contact resistance, a thin (≃ 30 nm) chromium layer is deposited on top of contacts before W deposit. Experimental results are shown in figure 8. For the same doping conditions the 1 µm contact resistance ratio between the Al and W systems is about 1/4 and 1/2 for n+ and p+ respectively in favor of tungsten system. For contact to polycide the resistances are equal. Let us note that this parasitic resistance gain for W compensates the larger resistance of W wires up to a few tenth of a millimeter. So, for short range interconnects W will provide better performance than Al.

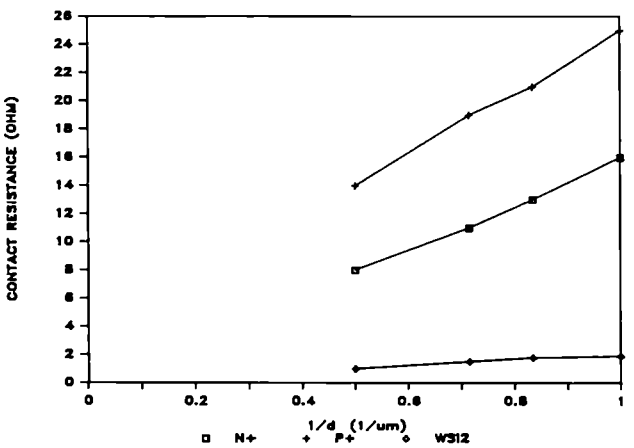


Fig 8 :

Variation of the contact resistance Vs 1/d, where d stands for the contact size, for contact of the Cr/W system to n^+, p^+ and WSi_2.

In comparing performances achieved with W or Al systems one must also take into account the electromigration effect. It is known that it limits the current density ; and therefore the maximum current which can be carried out in a wire for fixed design rules. Since, propagation delays depend on the parasitic capacitance to current ratio, the electromigration phenomenon in Al introduces a limit in propagation delay as drawn in figure 9. It is worth comparing this electromigration limit to the well known sheet resistance associated limit as done in figure 9. It appears that for short wires, up to 1 mm, refractory material such as sputtered tungsten with 0.4 Ω/sq leads to better propagation delay than aluminium. For medium length CVD tungsten with 0.1 - 0.2 Ω/sq is the best choice. Aluminium is found to be significantly better only for wires such as global busses.

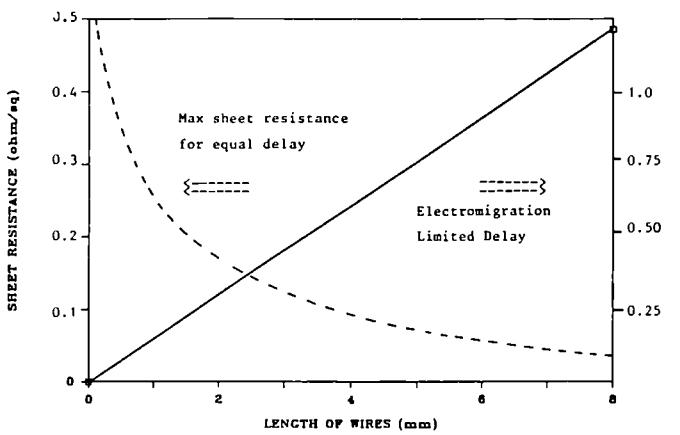


Fig 9 : Resistance and electromigration limited delay (after P.D. Chattergee interconnects (VLSI workshop 1985).

Finally, one can conclude that the optimum interconnect scheme will use W as a first level and Al for the second one. This will provide both process simplicity, and flexibility to designers.

For the 1 µm demo at CNET, both sputtered and CVD tungsten were investigated. In addition to the above mentionned advantages CVD has very good planarizing properties because it refills contact holes. However, the equipements available today are not industrial. So we decide to run the demo with sputterred W mainly ; a few lots of wafers with both materials will be prepared in order to assess CVD W on actual devices before to introduce it in the 0.7 µm process.

5. DEMONSTRATION

Before the end of 1987, SPECTRE partners should demonstrate : (i) that basic techniques suitable for future integration in a 0.7 µm process have been developped ; (ii) that devices with actual 1 µm electrical and design rules can be fabricated and (iii) that the pilot or manufacturing lines of the partners will be capable within the next two years of assembling and demonstrating a 0.7 µm process.

The first item (i) will be demonstrated indeed by the results described by the different tasks dealing with basic techniques. Moreover several of those techniques will be assembled in process demo as summarized in table 5.

Table 5 SPECTRE basic techniques to be demonstrated at :

	SGS	MHS	CNET
SUPERPLANOX with twin tub	X		
1 µm field isolation	X		X
1 µm transistor with LDD	X		X
Refractory gate		X	X
1 µm contact hole etching			X
Tungsten interconnects			X
dielectric planarization			X
Via opening with slope			X
Double Aluminium	X		

To assess the feasibility of the 1 µm electrical and design rules agreed between the partners (cf tables 2 and 3), CNET will process a demo circuit including :

- a whole set of test patterns designed in such a way that not only the design rules but also their sensitivity to process variations can be demonstrated.

- two 4 K SRAM with 6 transistors all designed by IMEC and CNET (this last being a reference circuit at CNET).

- an image processor (with about 80.000 transistors) designed by B.T.

At last, the capability of both SGS, MHS and CNET to process actual devices with complexity in the range 5.10^5 to 10^6 transistors could be proved by results they got in this range on proprietary circuits (which are no longer in the precompetitive field as defined for ESPRIT programs).

6. CONCLUSION

The first year (1985) of SPECTRE was mainly devoted to organize the work between so many partners. Cooperative work became effective in 1986 with a lot of basic techniques being investigated. This year we came to many conclusions which have been of significant help to each partner engaged in assembling its process. 1 µm CMOS, using some of the SPECTRE basic techniques has been debugged and validated by processing several lots of test pattern structures. The demo circuit will be launched in the very next month.

By the end of the year, at last, three of the SPECTRE partners i.e. SGS, MHS and CNET will have reached the 1 µm level in there pilot lines and will start with the assembling of the 0.7 µm process. The large quantity of technological or physical information and the process modules now available within the SPECTRE consortium will be of great help for that.

** The work described in this paper has been carried out by several SPECTRE partners. Much of the electrical results come from SGS and CNET.*

Project No. 824

A EUROPEAN PROGRAM ON WAFER SCALE INTEGRATION

Jacques TRILHE

THOMSON-SEMICONDUCTEURS
BP 217
38019 GRENOBLE CEDEX, FRANCE

Six European organizations : Thomson-Semiconducteurs (F), British Telecom (UK), Laboratoire d'Electronique et de Technologie de l'Informatique (F), Institut National Polytechnique de Grenoble (F), Technische Hoschule Darmstadt (FRG), National Microelectronic Research Centre (IR) are cooperating in a European program on WAFER SCALE INTEGRATION : Esprit 824. The main problem being addressed is the yield of a 100 cm2 device.

The first step has been the realization of a test mask in order to introduce the switches, necessary to discard faulty elements and to replace them by spares, into the Thomson 1.2 um CMOS technology. The switches can then be programmed either by laser or by e-beam (floating gate FETs). Three WSI products will then be designed and manufactured : a 4.5 Mbits static RAM, a systolic array for image processing and a 16 bit microprocessor tolerant to end of manufacturing defects.

The 4.5 Mbit static RAM is organized as 256 kwords of 18 bits. The basic cell used to build the RAM is the Thomson 64 Kword of 1 bit static RAM. Four cells, with an extra one among four decoder, have been put together to implement a 256 K block. The wafer scale memory will be powered with 5 volts and will have an access time of 100ns.

The systolic array has an SIMD architecture and the fundamental problem being addressed is the configuration of the wafer. A virtual array of 128 x 128 processors, each of which consists of a one bit adder and 128 bits of RAM, is to be built on the 4" wafer.

The 16 bits microprocessor tolerant to end of manufacturing defects is the first block in a family of defect tolerant building blocks. The goal is to build custom system on a wafer by putting together the microprocessor, ROM, RAM, PIA, ACIA, UART, ... and making customized interconnection with a sea of gates.

The program started in May 1986 for a duration of 4 years.

1. INTRODUCTION

It is clear that the increase in complexity of IC in the coming years will be obtained by the decrease of minimum features down to 1 or .8 µm and the increase of chip size (IBM claims 2 cm2).

In the nineties the rate of reduction of minimum features will decrease due to the fundamental limits of the silicon device and an increase in chip size, perhaps up to WSI level, will become a necessity.

The need for the increase in complexity of the component is economic. If we consider that packaging cost of a system is higher than marketing cost plus design cost plus silicon manufacturing cost plus testing cost, it becomes obvious that putting several dies in a single chip will drastically decrease the cost of a system.

In addition, the increase of integration goes with a decrease of pin-count, leading to an extra decrease in packaging cost ; as predicted by Landman and Russo Rent's rule (1971) |1|

This increase of integration leads to other major advantages of Wafer Scale Integration : higher speed due to shorter interconnection length and better reliability due to fewer inter-level interconnections.

The decrease of cost of a system while it is getting smaller, more powerful and more reliable is a general phenomenon : see for instance the Personal Computers. WSI will probably be the next step, at the hardware level, continuing this trend.

Possibilities of Wafer Scale Integration (WSI), are being evaluated in the form of a European project : ESPRIT 824. Partners involved in this project are British Telecom (UK), Laboratoire d'Electronique et de Technologie de l'Informatique (F), Institut National Polytechnique de Grenoble (F), Technische Hoschule Darmstadt (FRG), University College Cork (IR), with the leadership of Thomson-Semiconducteurs (F). The project started on May 15th, 1986 and will last four years. The first step has been the study of the various wafer scale products that have been announced |2| and the evaluation of products needed by industrial partners : "Thomson Branche Equipement et Système" and British Telecom. A test mask has been processed and tested by Thomson, LETI and Cork university to evaluate switches programmed by e-beam or laser and the compatibility of a field as large as a wafer with state of the art CMOS photolithography. Three WSI demonstrators are now being designed : a 4.5 Mbit static RAM, a 128 x 128 systolic array and a 16 bit defect tolerant microprocessor within Thomson, British Telecom, Darmstadt and Grenoble universities. A first batch is expected in early 1988 and a second run in the end of 1989. The last year of the project is dedicated to application to systems. The four parts of this paper will describe the test chip and the three demonstrators. We will firstly compare the three demonstrators to the worldwide work on WSI.

In the field of memories, NTT (J) was the first company to implement in 1980 a 4 Mbit ROM |3| (on a 3" wafer) for word recognition and a 1.5 Mbit SRAM on a 4" wafer in 1984 |4| |5|. Sinclair (UK) has demonstrated a .5 Mbyte dynamic serial memory on a 4" wafer for replacement of Winchester disk drive of portable computers. For this application, the announced access time of 10 µs is quite sufficient. A 7 Mbyte DRAM on a 5" wafer was announced |6|. Inova Microelectronic Corp. (USA) had a 4 Mbit SRAM project very similar to the one of ESPRIT 824. The main differences were a centralized decoder and a memory cell of 4 transistors. Therefore Inova at least doubles the cost of reticles and of photolithography and its memory cell is more sensitive to process tolerances than ESPRIT'one. Recently Inova changes its target and plan to use larger building blocks (256 K instead of 64 K) in order to produce, with Japanese process, a 24 Mbit part on a 5" wafer. Micron Technology (USA) is addressing the tricky problem of a 4 Mbit dynamic RAM. Considering that a memory cell of a DRAM has one transistor while a static RAM has four or six transistors the correspondance between the complexity of SRAM and DRAM can be established. We conclude therefore that the ESPRIT 824 memory is state of the art.

Processors interconnected only to near neighbours is also an application which is well suited to WSI. Pipe-line processors have been studied on paper at GTE (USA). A wafer scale systolic array has been validated on a small chip in North Carolina university (USA) |7| |8| |9| and an architecture of a string processor

has been proposed by Brunel university (UK), under an ALVEY grant. A 2-D array for Fast Fourier Transform has been realized in MIT (USA). The systolic array of ESPRIT 824 will probably be the first one to be implemented as a full wafer on silicon. A 128 x 128 array is expected, on a 4" wafer, many times the computational power of a NCR GAPP board of the same area.

The third ESPRIT 824 demonstrator attacks random logic circuits. It is more advanced research and will probably be a more widespread study in the world when three or four level metal technology is available in the industry.

2. TEST CHIP FOR WSI

Wafer Scale Integration will lead to zero yield if there is no possibility of discarding faulty elements and replacing them by spares. This point must be taken into account at the architecture and testability level as well as at the technology level. Our test mask is dedicated to the optimization of switches programmed either by e-beam or by laser. It can be advantageous for reconfiguring a wafer to have both switches that are normally on and switches that are normally off. If there are reversible, it is a major help for test. Copper tracking after passivation of the device is also being studied within project 824 and will be used to power the good parts of the wafer |10|.

2.1. Switches

2.1.1. Floating gate FETs

The floating gate FET is a very attractive switch for the following reasons :
- possibility of having a test configuration at end of manufacturing as normally on and normally off switches are possible for a CMOS process,
- increased testability : possibility of programming and erasing the switch as often as necessary,
- low cost : in a CMOS process, only an N channel depletion implant has to be added to the process,
- high density due to the small dimension of the active element. Additionally no extra pins are needed to control the switch (e-beam in a Scanning Electron Microscope, SEM, is used to charge the gate).

The only limitation is the low retention time in military conditions |11|. Floating gate FETs are illustrated on Fig. 1.

With e-beam irradiation, an N-channel depletion device is turned on and a P-channel enhancement device is turned off.
For 1nA incident beam, typically 120 μs is needed to charge a 7 x 2 um2 gate. For erasing, standard UV techniques can be used to eliminate the charge of the floating gate by photoinjection through the gate oxide. The UV can be either given by a flood lamp or localized with the use of a laser beam (in order to erase a single floating gate selectively).

It is more attractive to use an e-beam to erase the floating gate as programming and erasing can be done in the same machine (SEM).

For a low energy beam the secondary emission is higher than the primary emission leading to a discharge of the floating gate. At high energies the gate is discharged by electron injection through the oxide. Such high energies may however, lead in damage in the surounding junctions and gate oxides.

In our test chip, the width of the gate was varied from 1.2 to 25 μm in order to be able to drive different currents. The parasitic effects on neighbouring MOS devices can be measured for separation ranging from 1.5 to 20 μm.

For programming the floating gate FETs we will use an ISI/SS40 SEM connected to an ABT/IL200 beam control system.

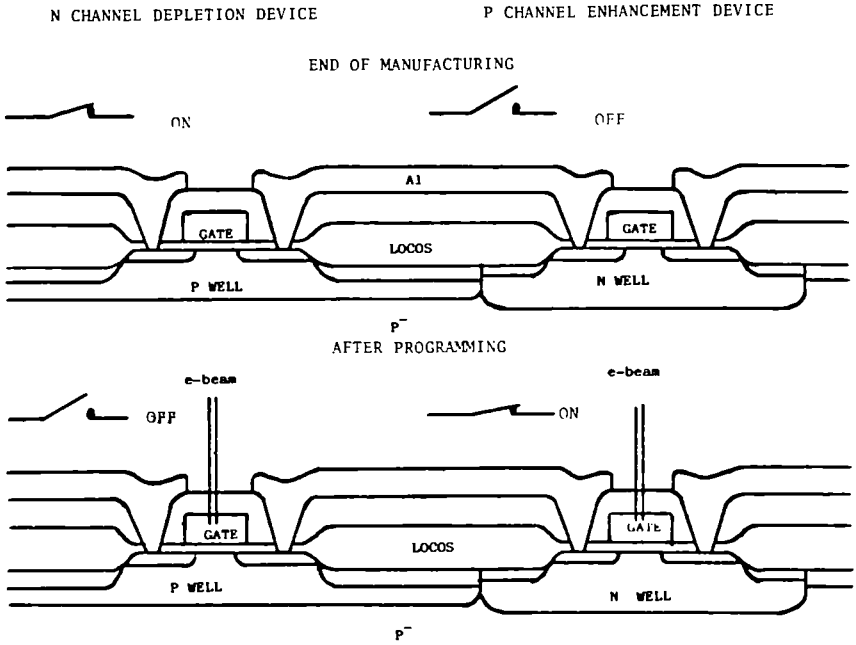


FIGURE 1
Schematic of operation of a floating gate FET.

2.1.2. Laser switches

Aluminium or polysilicon fuses are in widespread use in the Integrated Circuit (IC) business, in particular for memories. Using a laser switch for WSI requires a high reliability laser machine, due to the large number of flashes required on a wafer which have to be all at the correct location and at the right energy. In this project, we will use all the refinement of hardware and software of the ESI 8000C laser machine. The yield of the laser switches can be measured on our test mask on a 16 K aluminium ROM, each cell of the memory is programmed with a laser shot. The efficiency of the laser machine will be checked by the comparison of the result of test of the laser programmed 16 K ROM with the expected values assuming a high yield of the decoder.

There is no widespread technology available to connect with laser. MIT |12| has proposed melting a gap between two aluminiums tracks in order to obtain an electrical connection. They obtained resistance in the range of 1Ω. Another approach from MIT requires minor modifications of CMOS process. They add between the two layers of Al-Si-Cu a deposit of amorphous silicon and two very thin, 100 Å, silicon oxide barriers |13|. A laser shot on this structure makes a conduction between the two aluminium layers with a resistance $< 10\,\Omega$. Yields of 99,5% have been obtained for a 25 x 15 μm2 laser link.

The poly resistor-dopant redistribution method of Minuto et al. |14| gives a charge of resistance from 10 expl0 to a few kΩby heating (with a laser) an intrinsic polysilicon film between two N+ doped areas. Such a high on resistance is a problem for power lines and busses.

With our WSI test mak we will evaluate the technique of connection with direct laser writing on the wafer and lift off of a subsequent aluminium deposit. The resistance of this switch is typically the contact resistance to aluminium (1Ω).

2.2. Connection through the wafer

2.2.1. Aluminium lines running through the wafer

Our target is to use a state of the art, CMOS, 1.2 µm fabrication line to process our WSI parts. This means that we want to use a single reticle for each level, except for the last aluminium level which will be exposed with a reduction stepper for the central area of the wafer and with a 1:1 projection aligner for bonding pads on the periphery of the wafer. In this way, there will be little extra cost for WSI wafer, unlike the case if e-beam direct writing on the wafer has been used or if 4 reticles per level are used, as in MIT |13| or Brunel |15|.

2.2.2. Copper tracking

Copper tracking technology has been developed at the University College Cork (IR) within ESPRIT 544. A description of the process can be found in |10|. The copper tracking technique is a method in which low cost, high conductivity copper patterns are formed on top of the passivation layer of a semiconductor wafer.

The copper is deposited by electroplating through photoresist and is an additive process with resultant economical use of materials. Copper patterns of varying thickness can be deposited on the same wafer allowing different applications of the technique to be used simultaneously. Contact is made to the wafer metallization via contact windows in the flass passivation layer.
Typically, for power distribution, a 3 µm copper track is deposited leading to a factor of 5 decrease of resistivity in comparison to an aluminium line and with negligible increase in capacitance.

Copper tracking can also be used for discretionary wiring to repair a break in an aluminium line.

3. THE FIRST DEMONSTRATOR : A 4.5 Mbit STATIC RAM

The highly repetitive architecture of a static RAM makes it the easiest wafer scale product to implement. The initial goal of the 4.5 Mbit static RAM memory was to check the reliability of the switches mentionned above. The discussion with people building electronic systems (CIMSA-SINTRA) has indicated that the most useful format was a byte + a parity bit. Wa have therefore organized our memory in 2 x (8+1) bits. In doing so, the memory is not only a vehicle to evaluate the yield of our switches, but also a usable product.

The cell used is a Thomson memory, organized as 64 Kword of 1 bit. On a 4" wafer we will implement 256 Kwords of 18 bits, or 18 blocks of 256 x 1 bit (Fig. 2).

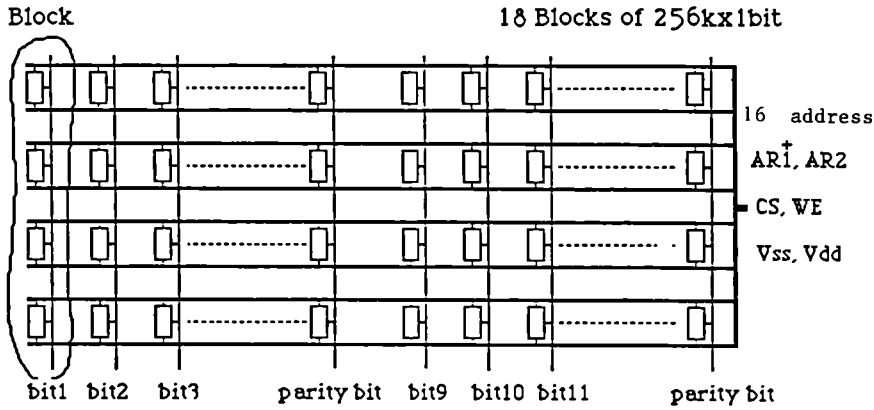


FIGURE 2
Virtual organization of the memory-WSI.

A block is defined by a set of 4 basic 64K cells. These 4 cells will be chosen to be as close as possible to eachother and will be connected to the same bit line (Fig. 3). This will optimize the use of connecting bit lines and their floor space on the wafer.

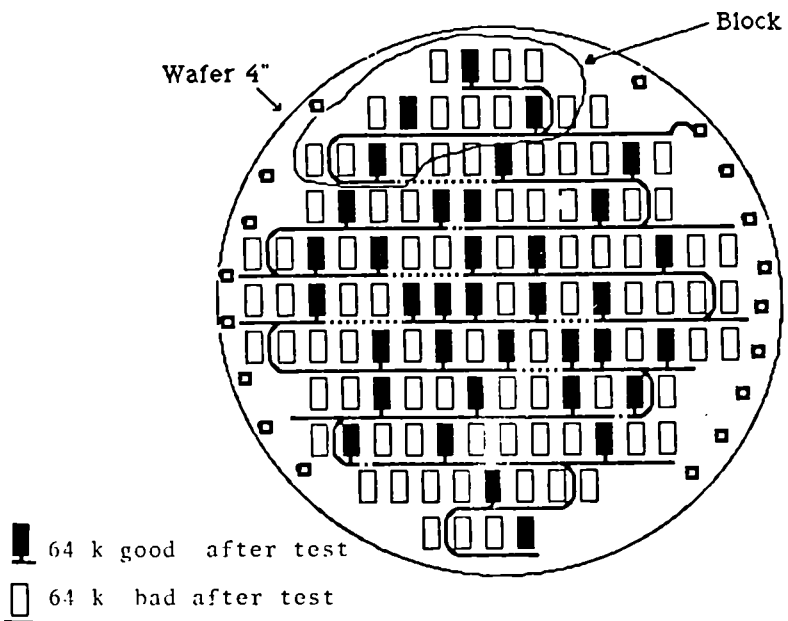


FIGURE 3
Reconfiguration model for Data network.

For each 64 K basic cell this organization requires a programmable decoder (1 among 4) which will be able to select one 64 K in a block of 256 K. This decoder powers only a quarter of the 4.5 Mbits (i.e. 64 K x 18 bits) during operating conditions reducing static and dynamic current. The decoders of the four 64 K of a same block are all different and are programmed with one of the switches described in part 1.

The interconnection network must provide : for each 64 K
. power supply : Vdd, Vss
. addresses : 16 lines
. write enable : 1 line
. cell selection : 2 addresses and chip selection.

The interconnection lattice must allow the connection and disconnection of cells (to discard faulty 64 K). It must also be reconfigurable to discard faulty connection lines. This will be achieves by the use of floating gate FETs and/or laser fuses and antifuses.

In addition 4 defects can be repaired in each 64 K cell by blowing fuses.

4. THE SECOND DEMONSTRATOR : A SYSTOLIC ARRAY

Systolic arrays are well suited to WSI because all communications are between nearest neighbours and thus connection length is very short with the integration on a wafer. Applications in the field of image processing in particular, are numerous.

BT has applications in the field of block matching |16|. When processing video images, in an attempt to reduce the amount of information to be transmitted to a receiving station, it would be nice to be able to spot blocks that had moved. Hence, it should be possible to define an arm as a block, and transmit that the arm had moved up, down, etc. This is impossible at the present time, but a first approximation to this is to split a picture into blocks (say 9 x 9 pixels) and try and predict where these blocks have moved to. To do this, a processor is required that can compute a 9 x 9 convolution for all positions in the search area.

This leads to a requirement for a processor that can compute approx 289 9 x 9 convolutions, for every position in the image in real time.

Thomson has applications in advanced display graphics. Thomson wish to be able to do geometric transformations on a high resolution colour image in real time. The actual problem is to be able to do a 4 x 4 interpolation on 1024 x 1024 12 bit pixels, in 1/30th of a second. In order words, it is necessary to be able to do 31.5 million 4 x 4 interpolations every second.

Our approach was to choose an architecture that is not only capable of solving these applications but will be of value for many other applications.

The rationale behind this was that the design, development and realization of any product as complex as a wafer scale array involves a level of investment and resource commitment that could only show a satisfactory return by being manufactured in relatively large volumes, and no single application was identified that would require high volumes.

Furthermore, the use of a circuit of this scale requires software compilation and emulation tools to manage and understand the complexity of the processing and data flow and these tools can aid the speed with which new problems can be tackled using general purpose hardware. The reasons for this approach are similar to those that fueled the development of microprocessors but the exis-

tence of a powerful general purpose parallel processing engine capable of performing at a rate of many thousands of MIPs opens up a whole new range of exciting applications. Our demonstrator is an SIMD machine, every processor performs the same operation at the same time. Communication is achieved by 4 way connectivity ; each PE connects to North, South, East and West neighbours. Each PE is a simple 1 bit adder/subtractor with 128 bits of local memory. A flag resister is included in the PE to enable local modification of the instruction according to the register content.

The configuration of a systolic array on a wafer raises many problems. All nearest neighbours of a cell may be faulty and yet a cell must be connected to the nearest good cell. The length of the wires is, therefore, longer than the minimal distance between 2 PEs. Of course, as the number of faulty PEs increases, the maximum wire length increases too. The connection length depends on the configuration method as well as on the yield. Therefore, among the many possible ways in which the good cells of the wafer can be connected to form a systolic array, some of them are more desirable than others. The PE has about 900 transistors, 84% of which are in the RAM. The area of the PE is approximately 0.2 mm2 in the 1.2 µm CMOS technology used. Reconfiguration at the PE level is, therefore, not attractive since the PE yield will already be very high. Reconfiguration will be achieved at the "chip" level (block of 8 x 9 PEs), using a decoder to discard faulty PEs.

5. THE THIRD DEMONSTRATOR : A MICROPROCESSOR BASED SYSTEM ON WAFER |17|

The goal is to integrate a complete dedicated system, usually including several printed circuit boards (PCBs), on a single chip by using the MegaPIL (*) approach.

The method consists of using pre-defined (and tailorable) blocks embedded in a flexible interconnection structure to implement a specific system. It is obvious that different applications such as signal processing or real time controllers cannot be implemented with the same set of blocks. The approach, therefore, consists in providing tailorable blocks adapted to different sets of applications. This means that, for each set of applications, a structure including all the blocks required by the appliction will be used. The global system uses a flexible connection structure which is implemented with a sea of gates. This allows both the realisation of interconnections between the blocks and the implementation of interface random logic which cannot be included inside the blocks (Fig. 6).

Since the goal is to integrate a microprocessor based system, the blocks that will be used are the usual microprocessor and peripheral circuits. The first block is the microprocessor itself.

Being the master block, the microprocessor has to be particularly adapted to the application, i.e. its instruction set must be defined according to the application specifications. It is obvious that in our approach, such a circuit cannot be designed as a full custom chip. This would take too long. A fast microprocessor design is based on fast design of its two blocks : the data processing part and the controller.

Since the circuit must be perfectly adapted to the application, an approach based on using standard bit-slice circuits such as 2901 and 2909 (as in the WSI Inc. products |19| cannot offer the specialized operators required by some applications. Additionally some area may be wasted because of unused capabilities of such blocks.

(*) MegaPIL is a trademark of Thomson Semiconducteurs.

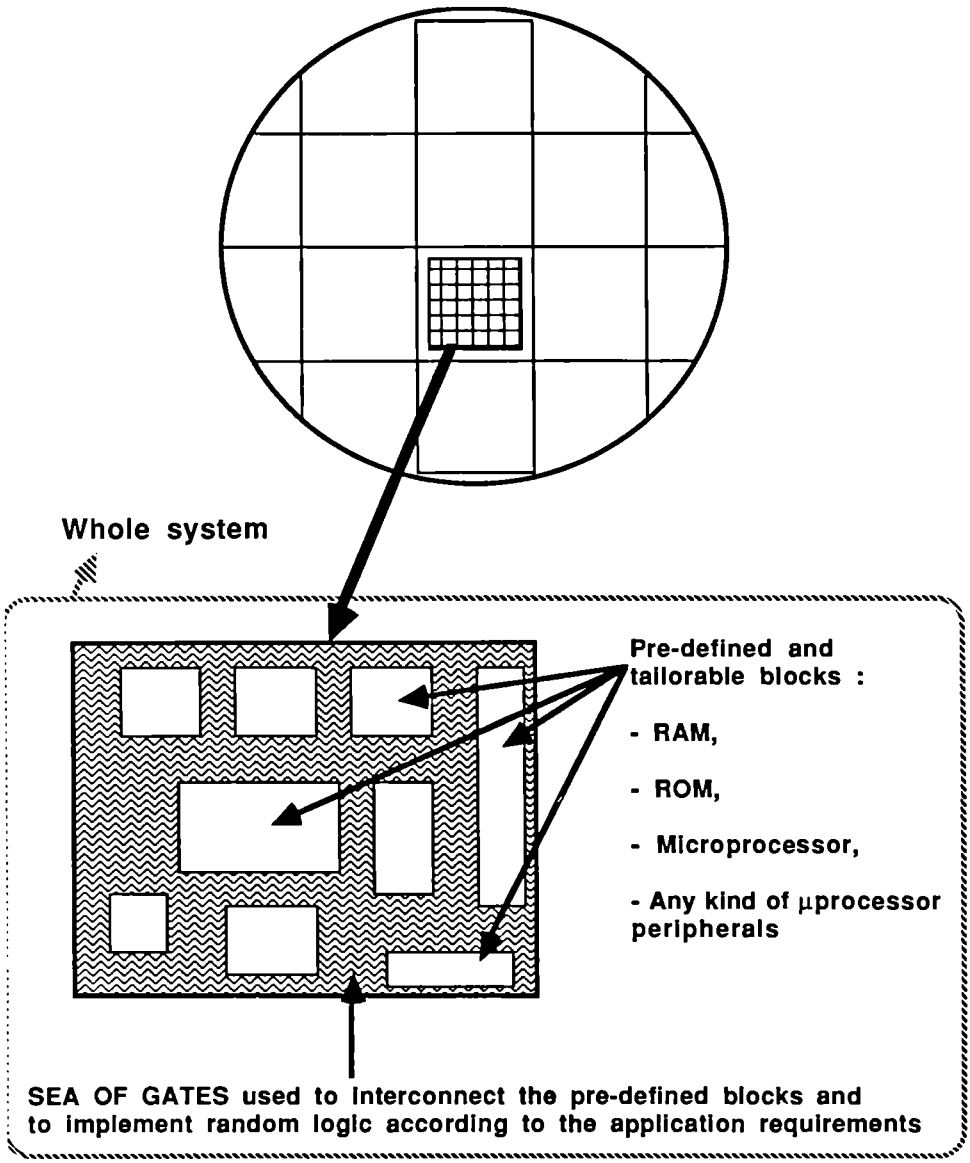


FIGURE 6
Example of hard blocks in a sea of gates.

Other blocks to be used will include the usual peripheral circuits for microprocessors. Several types of RAM and ROM memories must be available as blocks to implement a system. These blocks must be tailorable so that the system designer can choose exactly the memory size and access procedure he needs to fulfill his application requirements. A second set includes interface devices such as UART, DMA, A/D and D/A converters and so on. These circuits may be tailorable too by allowing the user to choose the number of I/O lines on a device, for example.

A third group consists of all the special operators such as arithmetical processors, specialized multipliers, etc.. These circuits are mainly unmodifiable and have to be used as they are.

The principle advantage of the MegaPIL approach is that the system size decreases from several PCBs to one chip. This has other advantages such as a reduction of the power consumtion because there are fewer buffers to power in the system. Another advantage is the reduction of connections between chips (and between boards) which implies fewer interconnects and wire bond and therefore fewer bad contacts. This improves the mechanical reliability by reducing the number of PCBs.

This is particularly important for systems working in harsh environments such as in planes, space, etc. These three first advantages are particularly interesting in the case of on-board systems where space, power consumption and reliability are critical points.

Also as a consequence of the reduction of interconnections lenght at the system level, the increased speed possibilities for chip-based systems are an important point for realizing powerful and compact computers.

Added to the fact that the design of very big circuits using new design tools may bring some changes to usual design methods, the yield of big chips is the most critical problem. The yield of large blocks with greater than 50 000 transistors will be low so that the final yield of a big chip made up of such blocks would be nearly zero if nothing is done to improve the situation. The way to improve the overall circuits yield, since the sea of gates yield is very good (because of the small cell size), is to improve the yield of "hard blocks".

It has been shown that reconfiguration must be performed at two levels [18] :
- big chip level : at this level, spare blocks are provided to replace those destroyed by big defects. This is mostly used for interface circuits which are generally smaller than microprocessors and memories. This requires adequate bypass and replacement strategies to be defined.
- block level : in order to raise the yield values for large blocks and to tolerate small defects, redundancy is provides within the block.

Since the biggest blocks are those requiring the most important yield increase we had focused our attention on their problems.

Two kinds of big block structures can be encountered : memories and microprocessor - like circuits. Reconfiguration of memories has been studied and used at an industrial level [10]. The reconfiguration strategy mainly consists in providing spare pages to replace faulty ones.

The microprocessor like blocks include a DPP and a controller. The reconfiguration of these two parts present different requirements and constraints but it is based on the same principle. The basic point is that one cannot easily correct more than one defect at a time. The circuit therefore has to be partitioned into sub-blocks where no more than one defect may appear. Starting from there, both the partitioning and the reconfiguration strategy differ for the DPP and for the controller. They have been presented previously [10].

Application example

In the following section, we present an application example which illustrates the above points. This application is an automatic railway control system. Its structure is shown in Fig. 7.

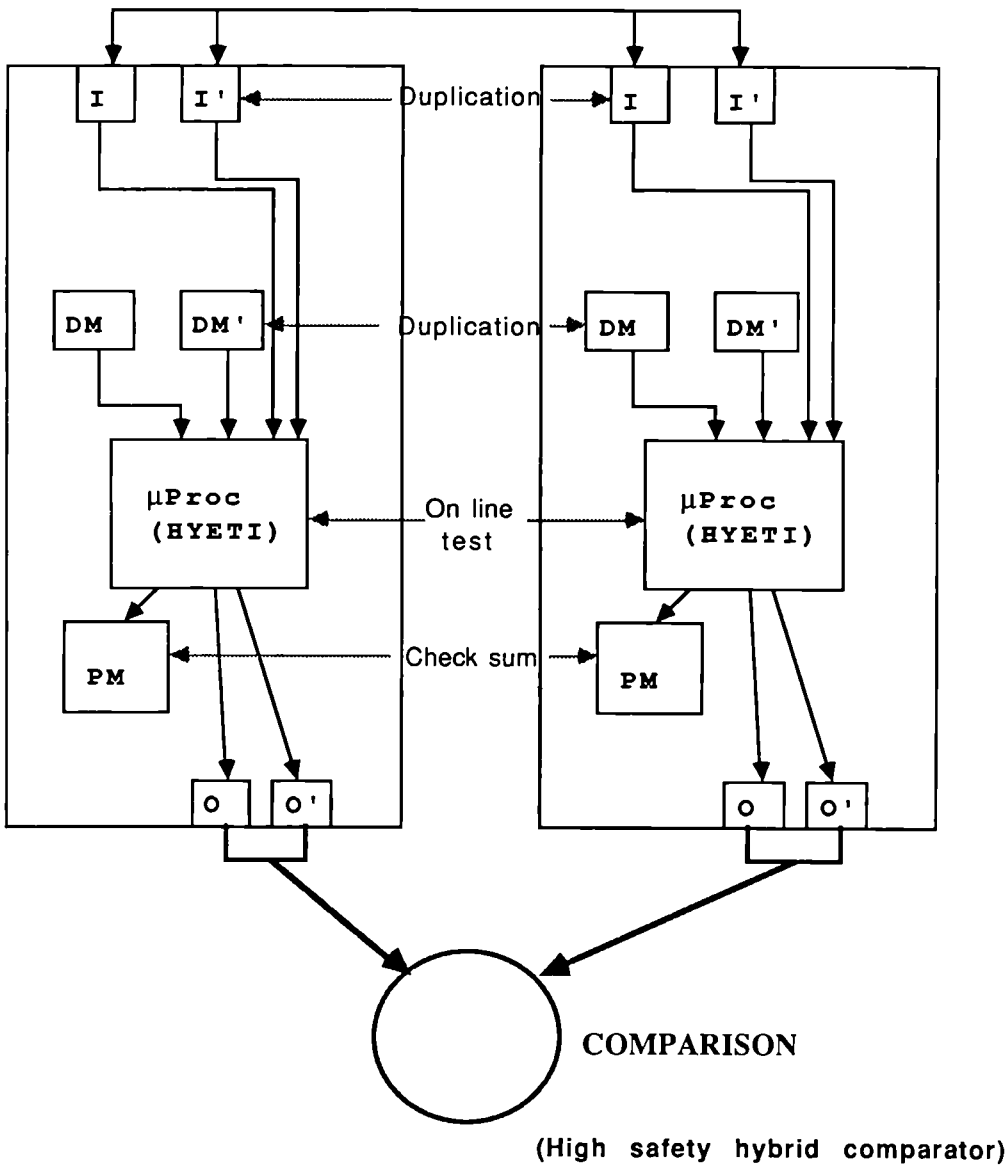


FIGURE 7
Automatic railway control system.

Realisation of such a system with conventional VLSI components requires 7 to 9 PCBs for input, 1 processor board, 1 security link boards on service link board. This involves the number of chips and the power consumption summarized in Table 1.

	Processor	Inputs	Outputs	Service link	Safety link
Boards No	1	7 to 9	4 to 6	1	1
ICs No in a board	75	42	72	35	40
Power consumption for 1 board (Watt)	≃ 9.5	≃ 5.5	≃ 9.5	≃ 4.5	≃ 5

TABLE 1
A possible realisation for the automatic railway control system.

The total number of ICs is 875. This leads to a 110 watt power consumption for an estimated 1.3 million logical gates system.

A big chip realisation of the system has the following features :
- it is a single chip system
- the type of pre-defined elements to be implemented allow several similar application to use this circuit (the application specific nature is taken into account when designing the inter-blocks connection)
- size : 50 mm x 50 mm
- power consumtpion reduced to less than 20 watts
- reduced cost
- fewer sources of failures : fewer PCBs, simple power and clock distributions

This clearly shows the interest of such an approach.

6. CONCLUSION

The ESPRIT project on wafer scale integration brings together efforts of six European partners to investigate three types of architectures which have a good chance of success : a memory, a systolic architecture, a microprocessor based system.

Strong cooperation between technology experts and designers has been the characteristic of the first step of the project. Starting from user defined products, companies and universities have put together design tools and methodologies to cope with the challenge of wafer scale integration.

ACKNOWLEDGEMENTS

We acknowledge CEC for support of this work under grant ESPRIT 824.
This paper is a compilation of results obtained through the cooperation of teams in British Telecom, LETI, Darmstadt University, Cork University, Grenoble University and Thomson.
Many thanks to all of them.

|1| Landman B.S. and R.L. Russo, December 1971, IEEE Trans. Compt., Vol. C-20
|2| Trilhe J., September 1986, Revue Technique Thomson-CSF, Vol. 18, N3
|3| Kitano K., S. Kodha, H. Kikuchi and S. Sakai, August 1980, IEEE Trans. Electron Devices, ED.
|4| Cohen C., January 1984, Electronics
|5| Veoka Y., C. Minagawa, M. Oka and A. Ishimoto, June 1984, IEEE J. Solid-State Circuits SC 19-3
|6| Jesshope C., W. Moore and A. Hilgen Eds., 1986, Wafer Scale Integration
|7| Heldund K., 1982, Thesis, Purdue university, Purdue, USA
|8| Hedlund K., July/August 1983, VLSI Design
|9| Hedlund K., 1985, proceedings ICCD
|10| Saucier G. and J. Trilhe Eds., 1986, Wafer Scale Integration, North-Holland
|11| Shaver D.C., February 1984, Solid-State Technology
|12| Asaitis J.Y., G. Chapman and J. Raffel, July 1982, IEEE Electron Device Letters, Vol. EDL 9. n° 7
|13| Wyatt P.W., J.I. Raffel, G.H. Chapman, B. Mathur, J.A. Burns and T.O. Herdon, 1984, proceedings of IEDM
|14| Minuto O. et al., October 1982, IEEE Journal of Solid State Circuits Vol. SC 17, n° 5
|15| Jones S.R., Warren K.D. and R.M. Lea, July 1986, proceedings of Silicon Design Conference, Wembley, UK
|16| Hein D., August 1984, IEEE Trans. on Electromagnetic Compatibility Vol. EMC 26, n° 3
|17| Hanria S., E. Dupont and G. Saucier, September 1986, proceedings of ICTC Conference, Limerick, Ireland
|18| Genestier P. and G. Saucier, september 1986, proceedings of ICTC Conference Limerik, Ireland
|19| WSI Inc. data sheets

Project No. 245

SOI MATERIALS AND PROCESSING TOWARDS 3D INTEGRATION

D.CHAPUIS - Y. GRIS - A. MONROY - E. MACKOWIAK - M. MONTIER
THOMSON SEMICONDUCTEURS, AVENUE DES MARTYRS, BP 217,
38019 GRENOBLE CEDEX, FRANCE

JL.REGOLINI - D. BENSAHEL
CENTRE NATIONAL D'ETUDES DE TELECOMMUNICATIONS, BP 98,
38243 MEYLAN, FRANCE

JL.MERMET - H. ACHARD - H. BONO - JP. JOLY
LABORATOIRE D'ELECTRONIQUE ET DE TECHNOLOGIES DE L'INFORMATIQUE, CEA,
GRENOBLE, FRANCE

L. KARAPIPERIS - G. GARRY - D. DIEUMEGARD
THOMSON LABORATOIRE DE RECHERCHE, THOMSON CSF, DOMAINE DE CORBEVILLE,
BP 10, 91401 ORSAY, FRANCE

KM. BARFOOT - M. FIELD - GF. HOPPER - DJ. GODFREY
GEC RESEARCH LTD, HIRST RESEARCH CENTRE, EAST LANE, WEMBLEY,
ENGLAND

DA. SMITH - DA. WILLIAMS - RA. MCMAHON - H. AHMED
MICROELECTRONICS RESEARCH LABORATORY, CAMBRIDGE UNIVERSITY, ENGLAND

CG. CAHILL - B. DUNNE - S. O'FLANAGAN - A. MATHEWSON - WA. LANE
NATIONAL MICROELECTRONICS RESEARCH CENTRE, CORK, IRELAND

ABSTRACT

Progress in materials development for the mezzanine 3D smart power demonstrator is reported. Significant improvements in the SOI starting material, especially the use of selective epitaxial growth of silicon in the seed windows, together with refinements of the laser and electron beam recrystallization systems has allowed the production of device grade SOI compatible with the requirements of the end of project demonstrator. High quality SOI devices have been produced. Fine geometry bulk CMOS devices were found to be essentially unaffected by subsequent thermal cycling required to achieve an overlay of SOI material formed by recrystallization. These device results confirm the viability of the demonstrator production technique. Full demonstrator device batches are currently in production.

INTRODUCTION

One area of the microelectronics industry which has attracted considerable research interest for a number of years is the concept of vertically stacking several planes of active devices for Three-Dimensional Integrated Circuits (1). While a number of means of achieving this have been suggested, the most common approach is to use stacked layers of energy beam recrystallized silicon (2), separated from each other and the bulk material by silicon dioxide. While an initial impression would suggest that the major benefits of such an approach would lie in packing density and speed performance it has been shown that the gains obtained in the context of a VLSI requirement are less than could be

expected (3) when the extra complexity involved in producing the devices is considered (4). However, the stacked SOI approach does offer significant advantages for the realisation of integrated, mixed technology systems, where separate layers of transistors can be individually optimised and separated by a high quality isolating oxide. In principle several varieties of radically different devices could be used in different levels of a 3D-SOI system, without seriously compromising the fabrication sequence of any of them. The layers could be built up sequentially with only the thermal load of subsequent processing steps affecting previous stages.

Two alternative configurations of such an approach, a fully stacked and mezzanine structure, are shown in figure 1. The layered nature is clearly shown, with the vertically stacked structure being seen as a desirable longer term goal toward which the mezzanine approach used here is a logical stepping stone. In addition, a polysilicon shield or ground plane level could also be integrated within the vertically stacked structure and is included as a diffused region in the bulk silicon layer for the mezzanine approach, thus providing improved DC and transient isolation for the control logic array. For the vertically stacked approach this ground plane has the added benefit of being a useful heat sink during top layer recrystallization and also a means of achieving greater planarisation.

In evaluating this concept and its realisation, Intelligent Power/Interface applications were defined as being particularly suited to the 3D-SOI structure. Specifically, the high quality dielectric isolation offered by the inter-layer oxide offers great potential in this application, in addition to the capability of building optimised bulk silicion power devices and high performance latch-up free CMOS SOI circuits (5).

In order to provide the greatest design flexibility in a demonstration of a 3D SOI intelligent power structure, gate array implementations were adopted for both bulk power devices and the SOI control logic. The technologies chosen for this work are a 50V lateral DMOS bulk technology together with a 3 um CMOS SOI process. These two processes are being combined to fabricate a small test-bed 3D-SOI gate array, suitable for semi-custom interfacing and medium current/ voltage (1A/50V) driving applications. Packaging constraints limit total power dissipation in a full sized version of this array to about 5 watts.

3D-SOI

Fully Stacked Gate Array

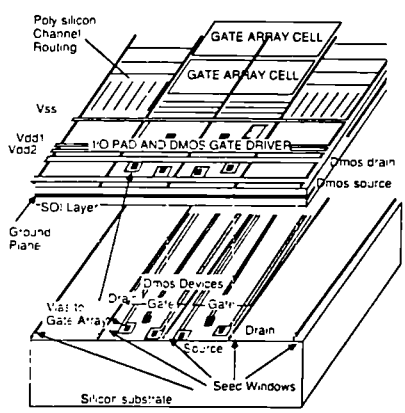


Figure 1.a

3D-SOI

Mezzanine Gate Array

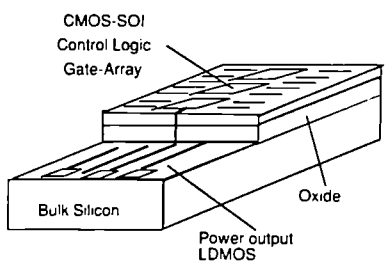


Figure 1.b

Recrystallization techniques which are particularly suited to the 3D-SOI approach are laser and electron beam zone-melting-recrystallization (ZMR) because of their localised, rapid heating of the silicon. In their simplest form these techniques produce large grain polysilicon with randomly oriented crystallites and grain boundaries, leading to a large scatter in the characteristics of MOS devices fabricated in the material. In this work both techniques are being used, with a 'seeded' approach adopted to provide bulk quality silicon in the SOI layer. The seed structure used in the test-bed design has a 43 um pitch ; however, some encouraging progress is being made with both the laser and e-beam in extending this distance. An improvement in approach this will clearly lead to greater packing densities and larger scope in the geometry and breakdown voltage of the power devices employed. Selective Epitaxial Growth (SEG) is used to planarise the seed structure and to reduce the mass transport that is currently observed as a feature of the ZMR approach.

The CMOS SOI portion of the array is designed so that the gate modules and routing channels are located entirely between seed windows ; this makes reasonably efficient use of the highly regular structure dictated by the seeding requirements. The design of these modules is more or less traditionnal (6) ; an example of a cell is shown in figure 2. It includes a device of each type (P and N), as two through cell routing vias. Both metal and contact levels will be programmable and a single level metal scheme is used.

A major consideration in the case of the bulk technology is that the devices must fit completely between seed windows to avoid destructif melting in the seed windows occurs during the ZMR step. This constraint on the size of the devices requires that a medium voltage (50-70V) DMOS technology be employed. This has been achieved by modifying the traditional circular geometry of the LDMOS transistor to provide a device of the requisite dimensions. These transistors have been produced in an n-well CMOS process which was modified to produce the necessary Vt and breakdown voltage characteristics.

Experiments have been performed on the LDMOS devices to establish the effects on the device characteristics of the recrystallization step and subsequent SOI processing. Initial results indicate that, provided the ZMR step is constrained to provide good quality seeded material without melting the bulk silicon under the isolating oxide (i.e is within the 3D power window), no detrimental effects are observed on bulk device performance.

This report will now consider in detail the materials improvements made by the individual laser and electron beam recrystallization and SEG groups. Device results are presented and the development of the end of the project demonstrator is reviewed.

GATE ARRAY

CMOS-SOI Cells

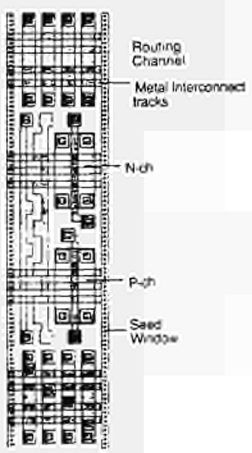


Figure 2

LASER RECRYSTALLIZED SOI

CNET-CNS

In laser Zone Melting Recrystallization (ZMR), two conditions have to be met if good quality SOI films are required : (i) give a well defined orientation to the recrystallized film, and(ii) avoid, as far as possible, the appearance of defects in the SOI region, or localize them in predefined areas. The first requirement is basically met by opening seeding windows in the underlying oxide in order to perform vertical epitaxy which then extends laterally over the isolating oxide. The second requirement has also been the subject of extensive studies : reflectivity modulation or heat-sink structures have been used in order to entrain the defects. The extension of the defect-free region depends on the entrainment technique, and/or on the behaviour of the seed during melting. However, the existence of voids in the SOI region adjacent to the seeding region and a non controllable mass-transport in the recrystallized film are the two major problems to overcome. The technique we are going to use for the future demonstrator of the project uses a filling of a periodic seeding network either by a selective silicon epitaxial growth or by a polysilicon deposition prior to the deposition of the polysilicon film to be recrystallized. Laser-ZMR is then performed using an elliptically focused spot of a cw-Argon laser. The elliptical spot allows the control of the solidification front during the successive partially overlapping scans of the laser spot accross the structure (7).

To demonstrate the feasibility of this process, <100>, 4 in. wafers are thermally oxidized up to 0.5 um. The oxide in then Reactively Ion Etched (RIE) in order to open 4 or 2 um wide seed windows with 16 µm periodic spacings. The RIE etching has been chosen since it gives steep angles in the seeding region. The seeding network is aligned with the <100> or <110> crystal orientations. In one set of samples, the seed window is filled up by the selective epitaxial growth (SEG) technique developed at Thomson/LCR (8). The selective epitaxial growth in the seed windows rises 0.3 um above the SiO2 surface level, and also extends laterally over the oxide by about the same amount.

In the second set of samples, a polysilicon layer is deposited and subsequently etched by RIE so as to leave the polysilicon only in the seed and on a sligh extension around it, i.e. as in the SEG process. Preparation of both sets of samples is completed with a 1 µm thick oxide as capping layer. The laser beam of a cw-Argon laser is focused in an elliptical spot giving a molten zone of about 300x100 μm^2 and is scanned over the wafer with its major axis at a slanted angle relative to the scan direction. The scanning speed is 10 cm/sec, the molten zone overlap is 50%, and the substrate temperature is kept in the 650-750°C range.

In laser-ZMR, the solidification front is controlled by the shape of the trailing edge of the spot. Moreover, when recrystallization proceeds at high speed, growth is expected to be faceted, and to exhibit (111) facets. In order to obtain a solidification front parallel to the <110> direction, the scanning has been oriented along the <100> direction. In figure 3, we present a sketch of the relative positions of the laser spot and the seed windows. In figure 4, we show a 2 µm seed, 40 µm pitch sample. The 2 µm seed region is filled by the SEG process. Seed windows are oriented in the <110> direction. The absence of interference fringes in the optical micrograph taken through an interferential filter indicates that no mass-transport has occured in these samples. The laser power window for 'good' recrystallization is about 10%.

The remaining defect in the SOI region is the so-called Principal Sub Grain Boundary (SGB) situated near one seed. Note that although the sample has been hardly chemically decorated, the SGB appears very fuzzy thus indicating a low misorientation. The large elliptical laser spot crosses several seed windows,

and when it scans two adjacent seed windows, two solidification fronts originate from the two seeds and meet together, resulting in a SGB. When the laser spot is scanned parallel to the seed windows with a slanted axis, the SGB is pushed towards one seed. In this work, a 30° angle has been chosen. No other defects (such as stacking faults, dislocations clusters, or twins) can be detected, at significant density, in the samples, and we obtain about 35 μm wide single-crystal SOI stripes across a whole 4 in wafer.

In the case of samples whose seeding areas are filled by polysilicon, the complete melting of the seed is needed in order to perform the epitaxial regrowth which requires a higher laser power. Therefore, the probability of producing and propagating residual defects is higher than in the SEG filling case : we encounter SGBs, stacking faults following the (111) planes (which may be due to a poor vertical epitaxy), and small voids. It should be noted that, this configuration is already optimal, which means that a <100> oriented seed and a laser scan identical to that employed with the SEG filled samples would have resulted in even poorer results (8). Since voids are due to the volumetric contraction of silicon when melting occurs in regions surrounded by solid silicon, the 30° spot configuration relative to a scan direction minimises the void formation. For a given configuration (nature of the capping layer, oxide and/or polysilicon thickness), the larger the thermal effect of the seed, the higher the occurrence of voids. So, the way to limit the void formation is either to have a narrow seed, or to cross it gradually, or to have an over thickness of silicon around the seed opening.

Transmission electron Microscopy studies have shown that the SOI stripes are free of defects (except the Principal SGB located near a seed). In the seed regions, the density of isolated dislocations is increased when the width of the seed increases ; the filling of the seed with poly-Si gives a higher density of defects than the SEG case. The occurrence of dislocations in the bulk under the seed region is higher with a poly-Si filling than with a SEG filling. Finally, the existence of dislocations inside the seed region must be related to the native oxide layer between the two silicon depositions. In future, great care has to be taken in order to minimise this deleterous effect (sample preparation before the poly-Si deposition after the SEG filling of the seed).

The good crystallographical properties of the SOI layer have been confirmed by electrical results obtained on 4 in. wafers of 40 um pitch, 2 um seeded samples filled by SEG. The detailed results will not be discussed here since they are similar to these presented by THOMSON on LETI wafers (in particular, the same technology is used in the CNET lab. for the two processes). Moreover, we will not discuss here several experiments made in order to test the recrystallization process on underlying processed substrates. The main conclusion for the material point of view is that we must take care in the design of the underlying substrate since it can act as a heat-sink for the upper level.

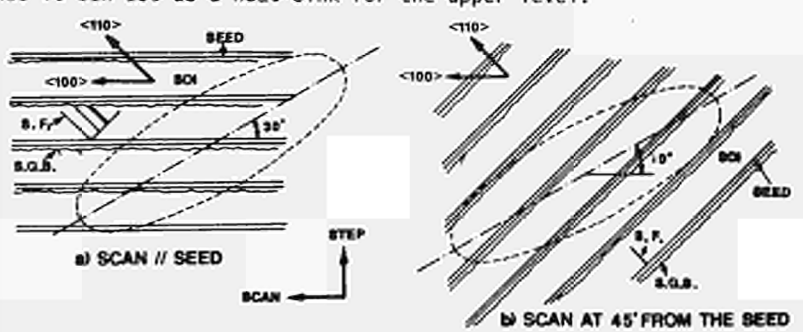


Fig. 3 : Schematics of the position of the laser spot relative to the scan stripes and seed orientations

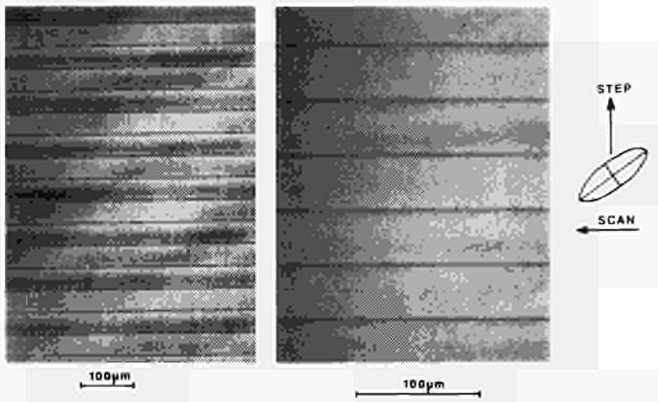


Fig. 4 : Optical micrograph of SOI chemically decorated sample, 2 um seed and 40 µm pitch.

In conclusion, 35 µm wide, defect-free SOI stripes have been obtained in laser-ZMR using periodic seeding. The filling of the seed by an over-thickness of polysilicon or monocrystalline silicon (obtained by a selective deposition process),together with the use of a controlled slanted elliptical laser beam, allows the production of defect-free stripes of SOI material over whole 4 in. wafers. Voids are eliminated and mass-transport is considerably limited. Finally, the filling of the seed by a selective growth process widens the laser power window, thereby lowering the importance of the laser power fluctuations. It allows also the use of a seed technique in 3D circuits where the upper levels have to be recrystallized without a too high heat transfer to the underlying circuits.

LASER RECRYSTALLISED SOI (continued)

CEA-LETI

The main points studies recently at LETI within the aim of the project are :

1 Low temperature process compatible with 3D integration

Many batches have been made with deposited oxide instead of thermal oxide as an isolation layer. We have demontrated that SOI recrystallization is fully compatible with such films.

The thicknesses (from 0.5 to 1 µm) of ZMR SOI layers are often not compatible with advanced CMOS SOI processes which need 0.2 to 0.35 µm thick ones. Accordingly thinning by oxidation is often needed which is not compatible with 3D integration. We have successfully developed low temperature RIE full sheet etching instead of oxidation with good control of the final thickness.

New low temperature processing steps (less than 900°C) have been developed and used in place of 3D incompatible ones (like gate oxidation, Si poly doping,...)

2 ZMR on thick oxide (> 1 µm) with a view to 3D integration

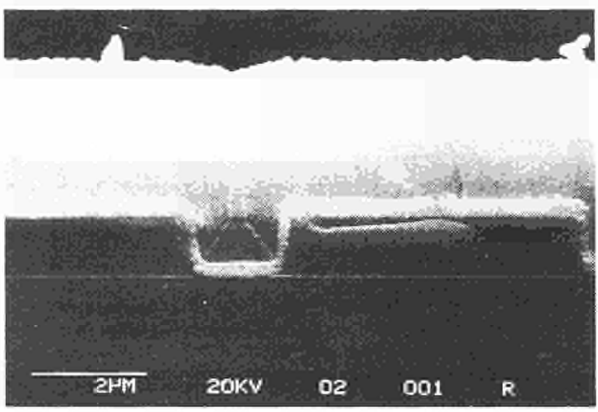
To obtain the compatibility of seeded recrystallization with thick oxide layers, as would be the case in 3D devices, we have worked in two directions.

We have used seed aperture filling using Selective Epitaxial Growth (SEG). (7) This technique improves the usable laser power window (from about 0.25 to 1 W). SEG filling allows good recrystallization in cases where classical seed windows lead to a reduction of the laser power window to zero.

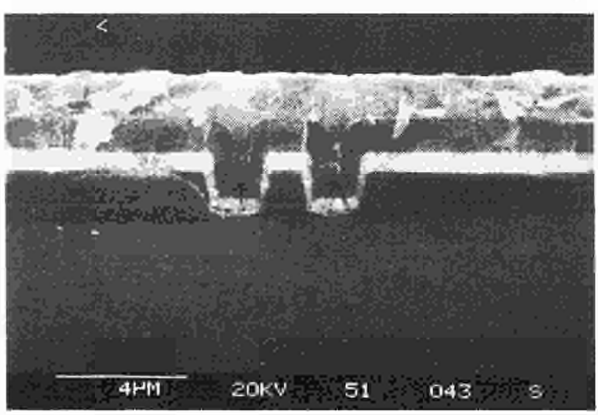
We have used in parallel a new seed design. It consists of discontinuous repeated seeds (perforation structure). By reducing the thermal impact of the seeds, this configuration allows the same improvements in the power window as those achieved by SEG using classical seed geometries but without the need of an extra step (9) (10).

3 Study of some problems linked to 3D integration

Recrystallization on metallisation lines made of tantalum silicide obtained by sputtering of a composite target, and compatibility of these lines with high temperature processes have been investigated. No severe morphological defects in the silicide are induced by recrystallization (fig. 5). The stability of the silicide in contact with the silicon in the contact holes at high temperature is the object of futher investigation.



reference



12 W

Fig.5 : Tantalium silicide lines after laser ZMR recrystallization

Moreover, wafers have been recrystallized on oxide layers of non-uniform thickness on the same wafer, simulating the thickness variations introduced by the bulk devices in 3D configuration. Some local modifications of the SOI layer thickness have been observed near the oxide steps due to mass transport effects during the recrystallization. New capping layers are presently being studied to reduce these problems as much as possible.

4 General improvements in the SOI layers

The discontinuous seed structure mentioned above allows one to obtain a greater area of high quality recrystallized material by reducing the stress of the recrystallized silicon near the melting interface. Using these seeds we are able to produce up to 100 um defect free materials at the center of the recrystallized stripes (fig. 6). This allows defect free areas of 60 µm width to be formed if we take into account the overlapping defects.

This configuration of seeds improves noticeably the mass flow effects which are observed in proximity to the seeds due to their thermal impact. This thickness variation is reduced to about 500 Å, instead of the 1500 Å variation which was sometimes obtained with continuous seeds. This variation becomes of the same order of magnitude as those due to the scan overlaps, these latter being of the order of 10% of the Si layer thickness.

5 ZMR attempts over bulk devices

We have demonstrated that recrystallization can be obtained on existing devices without any influence on their electrical characteristics. Two batches were recrystallized, one with seeds and the other without seeds. If dewetting can be avoided on such structures, the associated variation of thermal paths (variation of oxide thickness, presence of interconnection lines) induces mass transport effects and an early appearance of critical defects. Further improvements of 3D material are expected.

In parallel we have developed a stabilised experimental set up which can process about ten 4" wafers per hour with good uniformity and reproducibility. The adjustable elliptical spot size allows a good adaptation with different types of device structure.

Fig.6 : Top view of a recrystallized layer after defect decoration (secco etching) using discontinuous seed structure (oxide thickness 5600 Å). No subgrainboundaries can be seen expt at the close proximity of the seed aperture.

SOI Electrical characterization

Thomson SC

A test batch bas been achieved by THOMSON in order to develop a low temperature CMOS SOI process compatible with the requirements of the project demonstrator. The SOI layer was formed by the laser recrystallization technique developed by LETI and insulative layer was a 3000 Å thick thermal oxide.

Main features of the 3 µm CMOS SOI process was :
- max temperature of 950°C (for poly doping), 850°C glass reflow
- RIE etching for Si islands

Results of the statistical measurements performed on both natural and enhanced transistors are summarized in table 1 and 2.

TABLE I : NATURAL TRANSISTORS

Average values on 66 TMOS tested per wafer .

TEST STRUCTURE	PARAMETER	VALUE	UNIT
N-channel transistor W/L=30/30 (µm)	Threshold Voltage	-0.72	V
	Mobility	900	cm²/V.s
	Yield	80	%
	Leakage current Vg=-2V , Vd=5V	60	nA
N-channel Transistors with various W/L: 20/5 , 20/3 , 5/20 , 3/20	L	0.8	µm
	Z	0.3	µm
P-channel transistor W/L=30/30 (µm)	Threshold Voltage	-1.5	V
	Mobility	205	cm²/V.s
	Yield	80	%
	Leakage current Vg=0V , Vd=-5V	100	pA
P-channel transistors with various W/L : 20/5, 20/3 , 5/20 , 3/20	L	0.5	µm
	Z	1	µm

TABLE II : ENHANCED TRANSISTORS

Average values on 66 TMOS tested per wafer .

TEST STRUCTURE	PARAMETER	VALUE	UNIT
N-channel transistor W/L=30/30 (µm)	Threshold Voltage	0.5	V
	Mobility	700	cm²/V.s
	Yield	83	%
	Leakage current Vg=0V , Vd=5V	1	nA
N-channel transistor W/L=20/3 (µm)	Yield	76	%
	Leakage current Vg=0V , Vd=5V	60	µA
N-channel transistors with various W/L: 20/5, 20/3 , 5/20 , 3/20	L	0.9	µm
	Z	0.1	µm
P-channel transistor W/L=30/30 (µm)	Threshold Voltage	-1.85	V
	Mobility	180	cm²/V.s
	Yield	90	%
	Leakage current Vg=0V , Vd=-5V	40	pA
P-channel transistor W/L=20/3 (µm)	Yield	86	%
	Leakage current Vg=0V , Vd=-5V	80	pA
N-channel transistors with various W/L: 20/5, 20/3 , 5/20 , 3/20	L	0.6	µm
	Z	0.9	µm

Results on natural transistors suggest that a N-type (1.E16/cm) remaining doping level exists in the SOI layer and the too high natural N-channel mobility indicates some SOI film constraints.
Modifications of the process have been proposed in order to reajust the P-channel threshold voltage and the N-channel leakage current of enhanced transistors and are currently tested on a new test batch.
Speed performance were measured on 249 stages CMOS ring oscillators with a fan-out of 1. Propagation delays at 5V were 2.5 ns and 1 ns for 5 um and 3 um designed gate lengthes respectively with a yield of 53% and 68%.
These good results give the way of an optimised low temperature 3 um CMOS SOI process suitable for demonstrator fabrication.

E-beam recrystallized SOI

CU, GEC/HRC and NMRC

Improvements have been made to the SOI material by altering the structure, orientation and deposition conditions of the material before recrystallization. The polysilicon has been deposited under conditions which give epitaxial growth of silicon in the seed windows and polycrystalline material over the isolating oxide, resulting in a small increase in the usable 3D-compatible power window. There was also an improvement in material quality because the preparation process included a HCl gas pre-etch before silicon depositions, which removed residual oxide from the seed windows and so reduced the defect density in recrystallized material compared with material which had not undergone the etch. If the residual oxide is not removed, it can be carried along with the regrowth front and nucleate defects such as sub-grain boundaries and small twins over the isolating oxide. Investigations have also shown that when the seed windows are oriented parallel to a <100> direction instead of the usual <110> the seedable distance without defects is greater, as regrowth tends to occur parallel and perpendicular to seed windows, and <100> is a preferred growth direction under conditions where faceted growth is taking place.

The greatest improvement in the material quality has been achieved by using material in which the seed windows are filled by selectively epitaxial growth (SEG) of single crystal silicon, produced by THOMSON-LCR, before deposition of the polysilicon. This planarises both the polysilicon and the capping layers so that the preferred stress relief of the cap does not buckle it during recrystallization. The thickness variations in the recrystallized film can consequently be reduced from 10-15%, obtained with non-planarised SOI films, to <5%. The usable power window for this type of material is also greater than for material without SEG filling and a greater seedable distance is possible because the seed windows constrict the heat flow less and are thus more efficient, ensuring that lateral heat flow dominates over vertical heat flow.

Experiments also have been conducted with the aim of exploring structures and conditions for multiple SOI layers for three-dimensional integration. Two layers of polysilicon deposited over oxide have been recrystallised using the same bulk Si seeding window and with conditions chosen either to recrystallize both layers simultaneaously or to recrystallize only the upper layer of polysilicon. In another experiment a single layer of recrystallised silicon on insulator has been used as the seed for a second layer, which was recrystallized to give device quality seeded single crystal silicon (see fig.7)

In order to test the effect of e-beam and laser processing on underlying devices, fine geometry (1.5 um gate length) CMOS transistors were fabricated in the bulk silicon up to the contact hole etch step, using a well established process. Single-level SOI layers were then deposited on these wafers and

recrystallized using 3D-compatible conditions at a background temperature of 850°C. Subsequent to confirming that the silicon layer had been recrystallized, the SOI level was chemically removed and conventional processing of the bulk devices completed. It was found that the electron beam recrystallization caused an electrical channel length reduction of less than 0.1 μm. Other device parameters tested (e.g. VT, breakdown voltage and off state leakage) did not show major differences from controls. Figure 8 illustrates a typical result for leakage current as a function of electrical channel length for p-type devices.

Figure 7 : SEM Micrograph of double SOI structure in which the second SOI level is seeded from an already recrystallized lower SOI layer

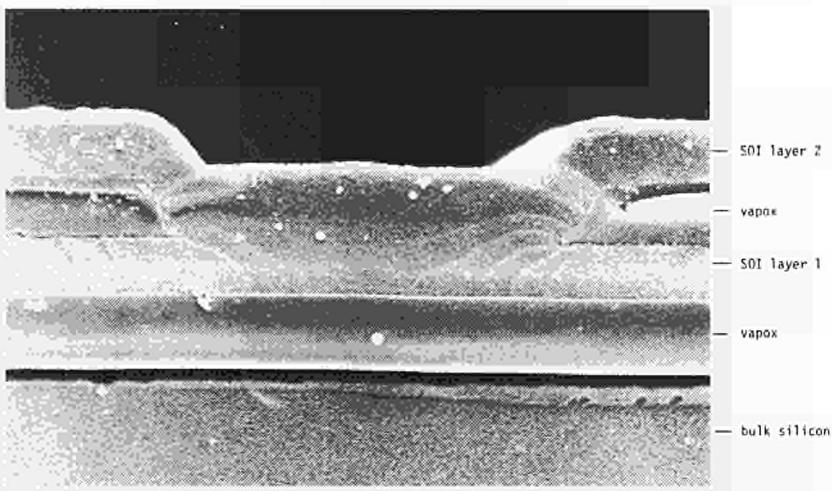


Figure 8 : Effect of electron beam heating on leakage current of bulk devices

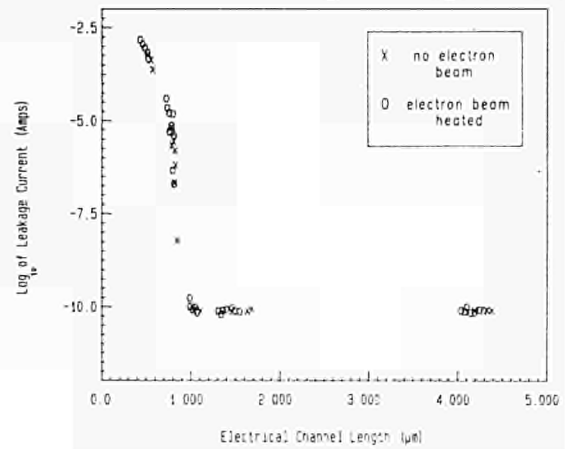


Table 3 Comparison of electrical characteristics of 10 x 10 μm SOI MOS transistors fabricated on thermal and deposited oxide isolating layers.

Device Type	Isolating Oxide	Electrical Parameter			
		Mobility ($cm^2V^{-1}s^{-1}$)	V_t (V)	Breakdown voltage+	I_{leak}* ($x10^{-11}A\mu m^{-1}$)
n-channel	thermal	465± 15	+0.95	+8.7	3.0
n-channel	APCVD	535± 20	+1.06	+8.0	3.5
p-channel	thermal	300± 25	-0.83	-12.5	3.0
p-channel	APCVD	220± 20	-0.92	-14.3	3.0

+Breakdown voltage at which a S/D current of 10 μA is reached for a gate bias of 1 volt below V_t.

*Leakage current, I_{leak}, for S/D bias of 5 volts and a gate bias of 1 volt below V_t.

In a separate experiment, fine geometry CMOS devices were fabricated in SOI layers recrystallised under e-beam conditions similar to the above. Two types of isolating oxides were studied ; thermally grown oxide which is conventionally used in single-level SOI work and atmospheric pressure chemical vapour deposited (APCVD) oxide which is used to avoid the high temperature process required in thermal oxidation. Electrical parameters obtained for such devices are presented in table 3. In this initial process, the electrical characteristics are encouragingly good with mobilities and off state leakages similar to those expected for a bulk silicon technology. In general, no substantial differences were found between devices produced on thermal oxides or deposited oxides, except for the case of channel mobilities. Whether this difference is inherent to the use of deposited oxide isolating layers or whether it is due to the recrystallization being performed under slighty different conditions for the two oxide types is currently under investigation.

In conclusion, it has been shown that good quality CMOS SOI circuits can be fabricated in material recrystalized under conditions which do not significantly affect the underlying fine geometry bulk CMOS devices. These results, together with the experiments showing successful recrystallization of multiple SOI layers, indicate the viability of true three-dimensional integrated circuits fabricated in material prepared by e-beam liquid phase lateral expitaxy.

Many improvements have been made to the Cambridge University electron beam system to make processing conditions more repeatable. Since the recrystallization depends critically on the performance of the line beam noise and ripple in the beam power have been reduced to less than ± 0.5 % from a worst case figure of ± 5%. The beam spot size (diameter) is now controlled to ± 3% and still further improvements are planned to achieve a target figure of ± 1%. Lateral jitter in the spot position has been reduced to less than the variations in spot size.

The electron source stability has been improved, partly as a result of changing from diffusion pumping to turbomolecular pumping of the system which gives a cleaner and a higher vaccum. Water cooled screens around the wafer have also helped to increase stability. Changes also have been made to the background

heating to improve the uniformity of the wafer temperature. This plays an important role in avoiding wafer warpage. The background temperature is set by a pyrometer control loop to ensure reproducible process conditions.

At the beginning of 1987, a dual electron beam system, based on research at Cambridge University, was installed at the GEC Hirst Research Center. The equipment is already playing a significant role within the project, both in terms of providing extra capacity for recrystallization of device wafers and in exploring new recrystrallization approaches to improving the quality and width of the single crystal SOI.

The developments in both wafer preparation and machine operation have permitted the precise determination of power windows for device quality recrystallized material and a consistent attainment of these power windows during operation. Material prepared by e-beam recrystallization has met in full the specifications laid down by the requirements of the demonstrator circuits at various preliminary stages.

SELECTIVE EPITAXIAL GROWTH IN THE SEED WINDOWS

THOMSON CSF

Practical 3D IC applications require insulating layers to electrically isolate the different circuit levels. The traditional silicon seeding approach has consisted of etching periodically spaced seeding windows in the insulating layer, and then depositing a uniformly thick poly-Si layer which follows the wafer relief structure. During recrystallization, the SOI layer is directly seeded from the substrate.

The combined requirement of fairly thick isolation layers (1 μm or more) and of rather narrow seed windows (2-3 um wide) resulted in steep, elevated steps over which the recrystallization front has to climb. This geometry causes significant mass flow problems, and the need to melt all the way to the substrate imposes a rather narrow power window for recrystallization. Melting into the substrate can also disturb the preexisting bulk devices.

The use of selective Epitaxial Growth (SEG) for filling up the seeding windows with monocrystalline Si plays a decisive role in resolving these problems, and in improving the crystalline quality, as well as increasing the width, of recrystallized stripes.

SEG can be performed by Chemical Vapour Deposition (CVD) either at Atmospheric Pressure (AP), or at Reduced Pressure (RP). For the present work, APCVD was adopted, but the feasibility of SEG by RPCVD has also been demonstrated in an experimental reactor (11).APCVD SEG is performed in the temperature range from 1000-1060°C, while RPCVD SEG can be performed at temperatures as low as 850°C.

The central problem in the SEG approach is the suppression of Si nucleation and growth on the insulator, while maintaining epitaxial growth in the seed windows. This can be achieved by adding an appropriate proportion of HCl gas in the carrier/source gas mix during growth. The APCVD work was done using a mixture of H2/SiH4/HCl gases in the proportion 100/0.6/0.8, respectively. Figure 9 shows a perfectly planarised structure achieved by this method. The excellent crystallographic quality of the selectively grown material was demonstrated by TEM studies (figure 10).

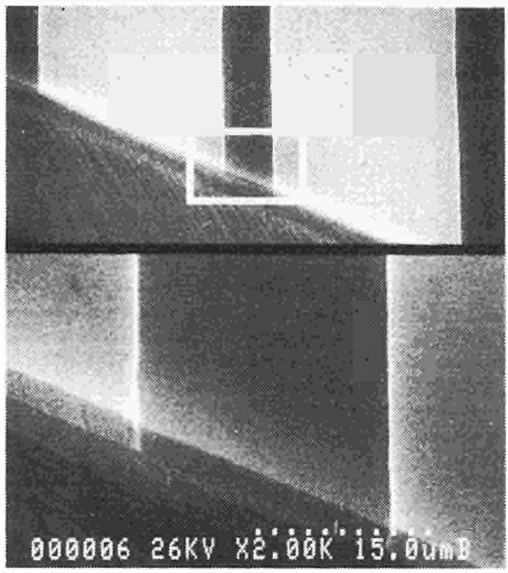


Fig. 9 : SEM micrograph of SEG filling of seed window
Upper part magnification : X 2000
Lower part magnification : X 10000

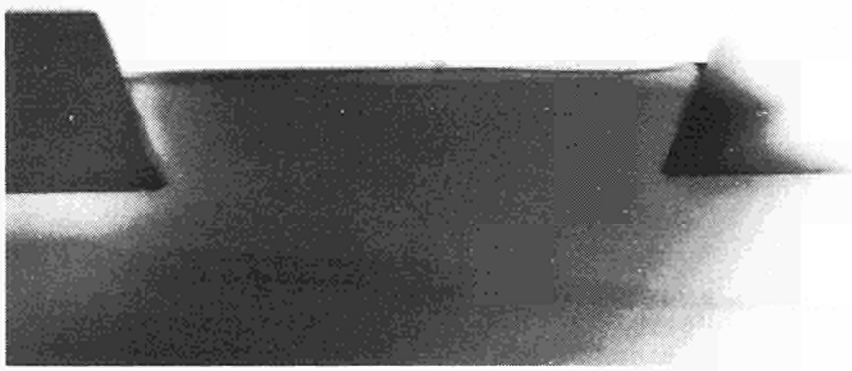


Fig. 10 : TEM cross-sectional view of SEG filling showing excellent crystallographic quality

The development of the project demonstrator

Technological orientations

It has been shown previously (3) that 3D SOI CMOS for VLSI is unlikely to provide sufficient benefit, in terms of packing density and circuit speed, to compete with single level technologies using bulk or SOI substrates. In contrast, the considerable interest in the development of silicon technologies

where devices of different types (e.g. CMOS, bipolar and power transistors) can be fabricated on a single chip, may provide an important application area for 3D SOI. Such mixed technologies are difficult to produce in a single level of silicon as the requirements of the different device types often conflict. However, using a 3D SOI approach, the development of a mixed technology with individual optimisation of the separate device levels can be envisaged.

In the light of these findings, the orientation of the current project and the end of year 3 demonstrator has been focused on the development of a "smart power" technology using a 3 µm CMOS SOI level to control medium current/voltage (1A/50V) LDMOS bulk transistors. The particular application considered is a stepper motor controller using a gate array design approach for both the CMOS and LDMOS levels. At this stage, a 'mezzanine' layout has been adopted where the SOI devices are displaced laterally from the underlying bulk devices. The bulk technology (apart from contact hole definition and metalisation) is fabricated before SOI recrystallization and CMOS processing. A single metalisation step is used for both LDMOS and CMOS devices. The CMOS SOI process used has been developed to minimise heat treatments which otherwise would result in unacceptable diffusion in the previously fabricated buld devices. The compatibility of the LDMOS process with the ZMR techniques has been demonstrated. It can be seen that the mezzanine smart power process produced within this project has been developed in a manner which will be consistent with a fully stacked structure.

Project Demonstrator Design

The detailed design of the mezzanine smart power stepper motor controller has been performed by NMRC and Thomson SC. A gate array approach has been used for both the CMOS and LDMOS levels to provide maximum flexibility of application. The layout of both levels has taken into account the constraints required by the laser and electron beam recrystallization approaches. For exemple, the CMOS devices has been placed away from the region of SOI where residual defects may occur and the LDMOS transistors have been laid out within the pitch of the seed windows required for SOI. Figure 11 shows a plot of the completed design illustrating the mezzanine approach and the overall size of the demonstrator chip. Also included on the demonstrator mask set are separate chips which will allow characterization of the LDMOS and CMOS devices.

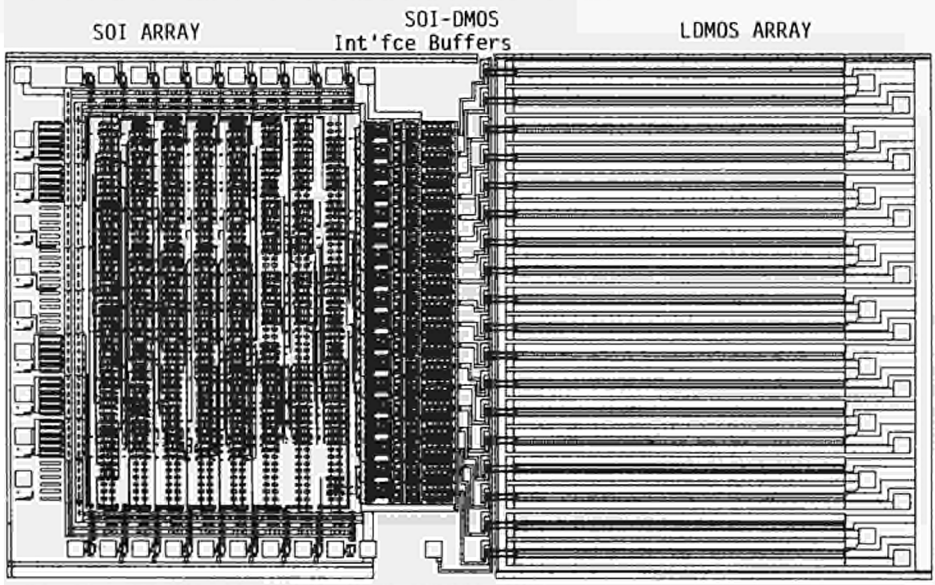


Fig.11 : 3D-SOI MEZZANINE GATE ARRAY LAYOUT

In detail, the stepper motor controller developed as the project demonstrator uses 432 basic SOI cells (846 transistors) arrayed in 8 lines and 54 columns with 27 input or input / output cells and 18 level shifters for the DMOS interface. The total area of the SOI level is 8.6 mm^2. The basic cell is composed of a pair of NMOS and PMOS transitors with separate gates (34 μm wide and 3 μm long) and 4 metal tracks used for intra-cell routing together with 2 polysilicon lines for short range interconnects. The level shifters allow the DMOS gates to be driven at 10-15V.

The high voltage bulk devices have been designed to fit between seed windows and this has been achieved by modifying the conventional circular geometry used for LDMOS technologies. The process used for the LMOS transistors has been developed from an N-well CMOS process to provide the necessary threshold voltage and breakdown voltage characteristics. The LDMOS array used in the demonstrator consists of 18 basic cells, each one composed of a pair of 2 mm wide LDMOS devices connected together to form a total area of 8.5 mm^2. In the 4 phase stepper motor application, 4 DMOS pairs are used to drive each phase.

The design described above has been completed and masks have been made. the demonstrator is currently being fabricated at LETI and NMRC. It is expected that first samples of the completed demonstrator will be available for electrical assessment in October 1987.

Conclusion

Over the last year, significant improvements have been made in both the structures of the SOI starting material and the laser and electron-beam recrystallisation systems. In particular, the use of selective epitaxial growth of silicon in the seed windows has increased the 3D-compatible power window and has significantly improved the quality and width (>40 μm) of the single-crystal SOI strip. The planarity of the recrystallised silicon ($<\pm 5\%$) obtained with such structures is much improved compared with that obtained with the conventional non-planar starting material.

Good progress in the production of the end of project demonstrator has been made. This takes the form of a 'mezzanine' smart power chip in which high voltage LDMOS bulk devices are controlled by an SOI CMOS logic level which is laterally and vertically offset from the underlying bulk power devices. All the design and mask making is complete and the bulk devices have been fabricated. Seed-window patterning will take place shortly and SEG deposition completed. Recrystallisation will then be performed at the laser and electron-beam laboratories. As has been discussed at length in the main text, all recrystallisation systems are now at a level where they can be expected to readily fulfill the requirements of the demonstrator. This has been verified both in terms of materials studies and measurements of devices produced under 3D-compatible conditions in either the bulk or SOI levels.

References

[1] M. Kanano IEDM 1984 pp 792-794
[2] e.g. Papers in Proc. symp on Laser and e-beam Processing of Electronic Materials eds. C.L. Anderson, G.A. Celler, G.A. Rozgoni. vol 80-1 ECS.
[3] M. Montier ESPRIT 86 Results and achievements pp 197-205 .
[4] S.L. Partridge IEDM 1986 pp 428-430 .
[5] W.A. Lane et al. IEE Colloquium on Intelligent Power Devices March 1987.
[6] S. Kazmi, M. Friedman GDN October 1984 p 173.
[7] J.L. Regolini, D. Dutartre, D. Bensahel, L. Karapiperis, G. Garry, D. Dieumegard Electronic Letters, 23 , 493 , (1987)
[8] J.L. Regolini, D. Bensahel, D.Dutartre, A. Peno, D.P. Vu, L. Karapiperis, MRS Europe, Strasbourg, June 1987.
[9] J.P. Joly, J.M. Hode, H. Achard, H. Bono, J.C. Castagner ECS Meeting 4-6 May 1986 Extended Abstract (Optimum conditions for Lateral Epitaxy in ZMR recrystallisation)
[10] S. Horita, H. Ishiwara Appl. Phys. Letters 50(12) (1987) p 748.
[11] L. Karapiperis, G. Garry, D. Dieumegard, Proceedings, 18th International Conference on Solid State Devices and Materials, August 1986, Tokyo, p. 713.

Project No. 574

NEW THREE-CHAMBER REACTIVE ION ETCHING SYSTEM MPE 3003

I. Hussla
Leybold-Heraeus GmbH, P.O.Box 1555, D-6450 Hanau 1, F.R.G.

H.-C. Scheer
Fraunhofer-Institut für Mikrostrukturtechnik
Dillenburger Strasse 53, D-1000 Berlin 33, F.R.G.

R. Smailes
UKAEA Harwell Laboratory, Oxfordshire, OX 11 ORA, U.K.

D.E. Webster
Johnson Matthey Chemicals, Orchard Rd, Royston, SG8 5HE, U.K.

A new three chamber reactive ion etching system MPE 3003 for 200 mm single wafer submicrometre pattern transfer in Al (Si, Cu) is described for the first time. Results on etch homogeneity and other important etch criteria are given as well as results on aluminium etch modelling regarding kinetics and chemistry. Pattern delineation for 0.4 µm features sizes by X-ray lithography with SiO_2 is also reported. This work was performed as part of the ongoing Esprit Project 574 "High resolution plasma etching in semiconductor technology: fundamentals, processing and equipment", a joint venture between European microelectronic processing experts, high-tech equipment manufacturers and a chemical materials specialist.

1. INTRODUCTION

In order to attain very large scale integration (VLSI), pattern transfer of 0.5 µm structures is required. X-ray lithography may be employed to achieve this and the results of adopting this approach are reported here. The other important step of pattern transfer is dry etching via reactive ion etching (RIE). High anisotropy, etchrate, selectivity, stability and reliability of the etch process are all needed. The etch apparatus should provide high etch yield due to high homogeneity of etching over the whole wafer area, small, or better no, contamination of wafer, little radiation damage, high system up-time and clean room compatibility.

A new multichamber etching system for Al (Si,Cu) etching of outstanding design and performance is presented, being a milestone in the ongoing project "High resolution plasma etching in semiconductor technology: fundamentals, processing, and equipment". A description of the system is given as well as first results with SiO_2 with respect to homogeneity and other important criteria, such as loss of critical dimension, selectivity and reliability. Preliminary results on aluminium etch modelling regarding kinetics and chemistry are presented. In the course of this work the RIE multistep etching approach is

also discussed in terms of its basic role in the realization of submicrometer pattern delineation. Aluminum etching in a three chamber apparatus is achieved with high throughput when two chambers are assigned to etching, while the third chamber is employed for anticorrosion treatment. Since the etching takes at least twice as long as the passivation step, a perfect match is provided.

2. SALIENT FEATURES AND DESCRIPTION OF THE THREE CHAMBER ETCHING SYSTEM MPE 3003 FOR AL (SI, CU)

The following are features of the three chamber system, with two for etching and one for anti-corrosion treatment:

- true 200 mm single wafer etcher
- high vacuum aluminum chamber with in-liner shields made of materials which are compatible with the etching process chemistry
- vacuum wafer transfer chamber for contamination control
- automatic pick and place vacuum wafer transport system providing wafer stress-free handling for either serial or parallel wafer processing
- cassette-to-cassette wafer vacuum handling
- 1.2 kW RF source capacitively coupled to bottom electrode of etching chamber with autotuning and RF matching (13.56 MHz source)
- anti-corrosion treatment (13.56 MHz RF source) upper electrode powered
- electrodes coolable and heatable over a wide range
- temperature controlled (up to 50°C) 4 channel process gas control with safety "fool-proof" gas box closed and nitrogen purged
- gas inlet via showerhead in upper electrode
- apparatus totally computer controlled, process status monitoring, SECS II protocol features
- smallest footprint (140x135 cm), clean room class 1 compatible installation

Figure 1 is a pictorial view of the three-chamber etching system MPE 3003. The etching system is modularly constructed and consists of: vacuum wafer elevator module, vacuum wafer manipulator module, and plasma etching reactor module.

In Figure 2 the cross-section of the etching system is shown. The wafer manipulator chamber is situated in the centre. This allows connection of six different chambers via perfluoroelastomer sealed vacuum valves. Residual gas pressures for the elevator, wafer and reactor are 10^{-2}, 10^{-3}, 10^{-4} mbar respectively.

The passivation step takes place without any further risk of ion damage to the etch substrate because plasma mode is employed when the upper electrode is powered. The development of an etch process for Al (Si, Cu) which is patterned by temperature sensitive x-ray photoresist is a delicate task. Therefore, fluoroptometry [1] has been employed for in-situ temperature measurement in order to avoid resist degradation. On the other hand, a certain temperature at the wafer surface will be required; this is induced by the sputter interaction what is necessary to promote desorption of involatile copper species.

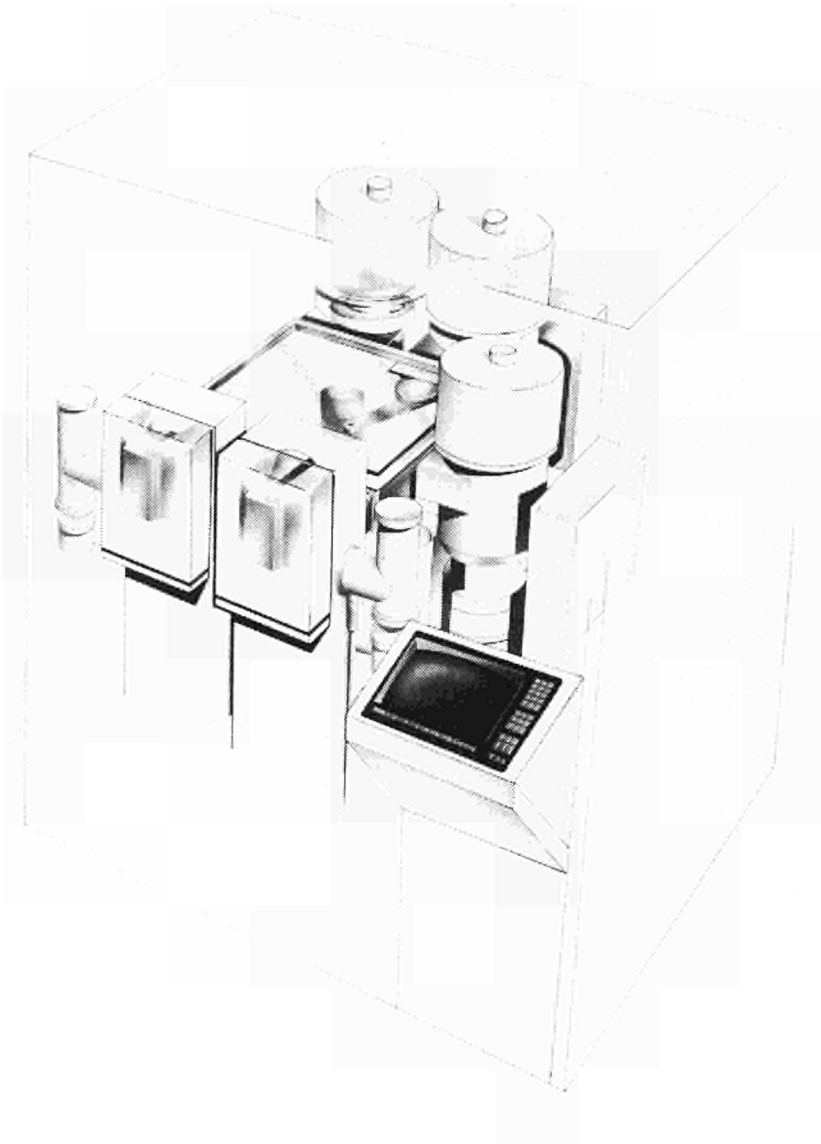


FIGURE 1
Schematic of MPE 3003, three-chamber RIE etching system

The computer control system of the MPE 3003 has been designed very compactly in order to minimize the footprint of the system. The control cabinet can be placed on the left or right hand side of the machine, elsewhere in the grey room or even in the basement together with the vacuum pumps. The operator terminal is integrated in the machine on the clean room side. A second terminal can be installed in the grey room. The etcher is controlled by means of an INTEL 8086 based system having a soft touch dust-sealed keyboard. Extended use of softkeys and text menus results in easy operation. Parameter input and service mode are locked by keyswitches. The control of the multi-chamber etcher can be adjusted for maximum wafer throughput, e.g. serial or parallel flow of the wafers through the chambers or mixed serial/parallel flow. A total of fifteen processes can be linked together in each process chamber including complete changes of gas chemistry, changes of various parameter settings such as gas flow, total pressure, electrode temperature, RF power applied and over-etch processing. During operation of the machine, the important process variables are displayed and monitored with respect to presettable limits, to ensure stable operation of the etcher. Endpoint detection is carried out by means of optical emission spectroscopy. Several security and alarm features are installed in order to guarantee high uptime and reliability of operation during production.

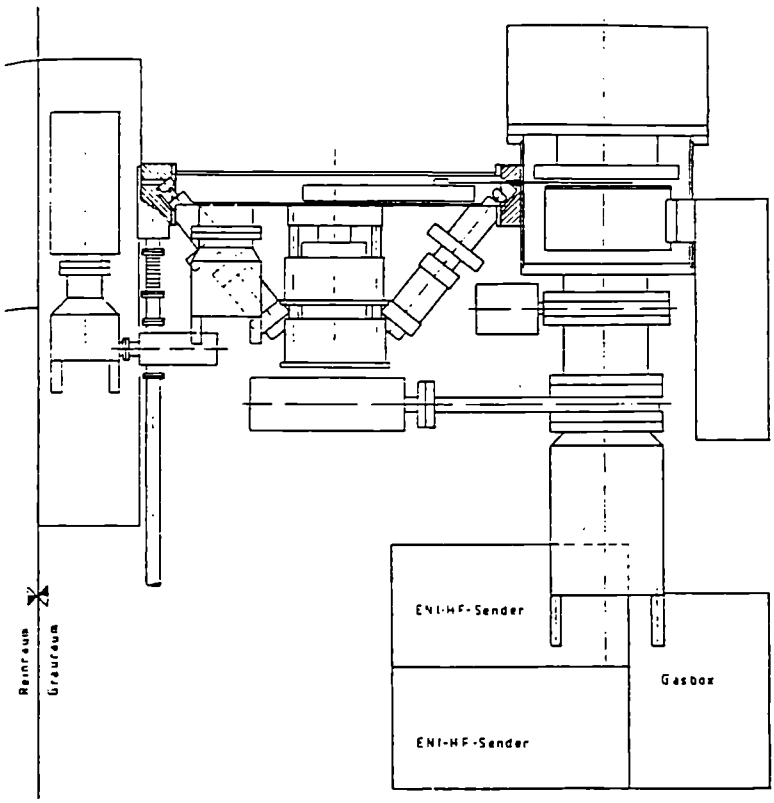


FIGURE 2
MPE 3003, cross section, wafer transfer chamber (middle), elevator (left hand side) and reactor (right hand side).

3. MULTISTEP RIE PROCESSES

Multistep etch processing offers the possibility of tailoring solutions to different etch criteria, such as etch rate, selectivity, or ion-damage, by a variation of process parameters. This variation may be carried out by computer control during a single wafer etch if there is no dramatic change (like changing chemistries), otherwise cross-contamination of different chemistries or delays due to gas exchange have to be taken into account. In former times, batch processing had a low etch rate, and it was possible to tolerate time consuming pumping or adjusting of a completely new gas mixture. In the single wafer etching approach for VLSI however, new criteria are applied. Here, total process control and reliability of the etch process by lowest contamination and therefore highest yield are required. This has led to the development of multistep etching via a multi-chamber approach.

Potential applications of this approach are listed here for a three chamber system. In the first chamber: preparation for etching, such as descumming, plasma hardening of resist and pre-etch of undesired oxides. The wafer is then transferred to chamber II for further etching via a pick-and-place vacuum manipulator. Here, the main etch process is carried out using fluorine or chlorine based chemistry. In the third chamber, post-etch processes, such as anti corrosion treatment or dry resist stripping may be performed. Particular examples where the use of multi-step process is required, are as follows [2]:

Etching of Poly-Silicon. First step: anisotropic etch with high rate, low selectivity to SiO_2. Second step: low anisotropy, but high selectivity to a thin SiO_2 layer.

Etching of Polycides. First step: anisotropic silicide etch. Second step: anisotropic etch of Poly-Silicon with high selectivity to SiO_2.

Etching of SiO_2-contact holes. First step: high etch rate, but low selectivity to Si. Second step: low SiO_2 etch rate, but high selectivity to Si. Multiple step and repeat with resist burning to achieve tapering.

Deep trench etch in monocrystalline Silicon. First step: trench etching. Second step: "clearing" of undesired features, such as trenching.

Planarization. First step: etch of resist until parts of SiO_2 layer are exposed. Second step: etch of SiO_2 and resist with respect to loading.

Tri-level lithography. First step: etch of an organic mask. Second step: etch of organic bottom layer.

Etching of Al and Al alloys for interconnect layers in IC's. First step: Etch of the Al/alloy layer. Second step: subsequent passivation. Third step: stripping of resist without breaking vacuum.

4. RESULTS

The MPE 3003 is specially designed for aluminium multi-step processing. We present in the following, principles and diagnoses of aluminium etching. We then report the performance of the RIE chamber with respect to bias voltage with respect to electrode gap and power applied to a nitrogen plasma.

Al and Al alloys are used as interconnect layers in integrated circuits. Fine-line aluminium pattern delineation requires a dry etching process with the following characteristics: high etch rate, good selectivity with respect to SiO_2, anisotropic etching to maximize the cross-sectional area of the metal leads, minimum resist erosion to ensure good pattern transfer fidelity, high uniformity and the ability to overetch in order to remove residues on the underlying substrate.

4.1. Diagnostics and Modelling

Optical emission spectroscopy (OES) is now an established method of monitoring plasma composition directly [3, 4], with the advantages of data acquisition occurring non-invasively and in real time. Collected spectra can be used to "fingerprint" particular etch processes under normal operating conditions and deviations from the fingerprint may be used to identify problems such as air leaks, high moisture content, impure process gases, or chamber contamination. OES can also monitor the concentrations of individual species in the plasma and their time dependence. This approach has been adopted under a variety of etching conditions to yield mechanistic insights into the aluminium etch process [5, 6], as well as acting as an end-point detector [7, 8].

OES with argon actinometry [9] was used to examine the complex interactions between plasma chemistry and machine parameters (flow, pressure and power) for a BCl_3, 5 % Ar discharge in the presence and absence of aluminium. Insight gained by work of this nature has allowed us to proceed to more complex gas mixtures as well as to optimize the design of the MPE 3003.

The BCl_3 plasma produced emission bands from B, BCl, and Cl. It was found that corrected emission intensities from B, BCl and Cl were all independent of flow rate over the experimental range, indicating that the equilibrium between production and removal is established rapidly relative to gas residence time in the chamber (typically 0,5 s). Emissions from B and Cl did not vary markedly with pressure. Over the range 25 - 150 mTorr Cl was approximately constant, while B decreased by some 40 %. The relative concentration of BCl showed a linear increase with pressure. At the same time, the corrected BCl signal intensity was independent of power, but the B and Cl signals increased approximately linearly with increasing power.

To account for these dependencies qualitatively, simple steady state models can be constructed in which production and removal rates are equated for the various free radicals and atoms present, with no account taken of the spatial variation of species concentrations. (The plasma emission is viewed in the central part of the discharge and not measured spatially.) BCl_3 is dissociated by electron impact to form radicals, which themselves can also undergo further electron impact

fragmentation. The rates of these processes are assumed to increase with absorbed RF power. For example:

BCl_3 ---> BCl_2 + Cl (1)
BCl_2 ---> BCl + Cl (2)
BCl ---> B + Cl (3)
B, Cl ---> loss processes (4)

Steady state treatment of the radical and atomic intermediates (BCl_2, BCl, B, and Cl) predicts that

(i) B and Cl should increase with increasing power, whilst BCl is constant (as experimentally seen)

(ii) that BCl should increase with increasing pressure, whilst B and Cl are constant only if the rates of atom loss (process 4) are pressure (i.e. BCl_3) dependent.

Two sources of atom loss can be identified - gas phase or surface. For the former a pressure dependent (third body) atom recombination process is unable to explain the results because the rates for such processes are likely to be several orders of magnitude too slow. B atom loss by gas phase bimolecular reaction with BCl_3 could account partly for the observations but Cl atom loss by a similar route would be a highly endothermic and hence improbable process. For a surface removal rate of atoms to be pressure dependent, a possible mechanism would involve reaction with adsorbed species, the fraction of adsorbed sites being dependent on BCl_3 pressure.

Considering now the influence of aluminium on the variations in corrected B, BCl and Cl signals, it is found that the same general dependence on machine parameters exists as in its absence. However, the relative concentrations of B and Cl were depressed in the presence of an aluminium surface, whereas the relative concentration of BCl increased. Emission lines from Al and AlCl were also monitored as machine parameters were changed. These showed similar variations with pressure and power as for B and BCl respectively, namely that the Al emission intensity was relatively insensitive to BCl_3 pressure but increased with RF power and AlCl emission was approximately constant with RF power and increased with BCl_3 pressure. The behaviour of B, BCl and Cl in the presence of the Al surface could be envisaged as due to reaction of B atoms with Cl chemisorbed on Al. Without knowing the identities of the surface decomposition products it is difficult to provide an unequivocal mechanism to account for the behaviour of Al and AlCl with machine parameters, as the etching mechanism of Al in a pure BCl_3 plasma is unknown. These species could be produced from AlCl, if this is the desorbed product (note the similarity in behaviour of the Al/B and AlCl/BCl ratios with pressure and power) but direct formation of AlCl (and hence Al by electron impact) from surface reactions is also consistent with our results.

4.2. Results on high resolution RIE [10]

The performance of a RIE process has been demonstrated at a SiO_2/Si layer interface. A highly reproducible and uniform silicon dioxide etch process based on CF_4/CHF_3 chemistry shows a resolution better than 0.4μm. In figures 3 and 4 the process is illustrated by a pattern transfer without any measureable

linewidth loss into a silicon dioxode layer of 0.6µm thickness. The sidewalls are vertical and the selectivity to the silicon substrate and the resist mask is excellent.

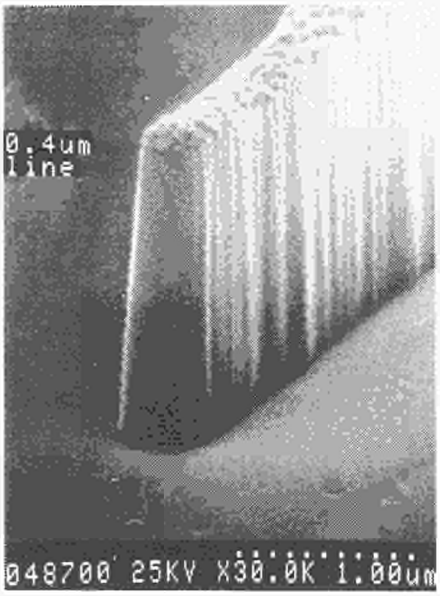


FIGURE 3
Photo of SiO_2 structure with resist on the top. Process parameters: CF_4/CHF_3-ratio: 3:5, pressure 8 Pa, power-density: 0.35 W/cm², SiO_2-etchrate: 80 nm/min.

FIGURE 4
Photo of 0.3 µm SiO_2 line after resist stripping. Process parameters see Figure 3.

Process control for in-situ etch rate measurements was performed by optical interferometry (endpoint detection, Leybold-Heraeus OMS 2000) or with an optical emission spectrometer (Monochromator BM 25, RCA Photomultiplier). Down to an exposed area of 1% of the wafer (e.g. contact hole etching) a significant change of the emission signal (CO line 483.5 nm) could be determined due to the high uniformity of the etch process. The SiO_2/Si etch selectivity can be influenced by variation of the CF_4/CHF_3 gas mixture. High etch rate ratios up to 20:1 are achievable by increasing the CHF_3 amount in the gas mixture but at the cost of increasing the polymer formation. With a 3:5 $CF_4:CHF_3$ gas mixture the selectivity can be controlled at 15:1. The polymer formation is very low, thus guaranteeing high uptime of the system.

We have characterised the MPE 3003 performance regarding bias voltage as a function of electrode gap and applied power for a nitrogen plasma as shown in Figure 5. Fitting of the measured DC bias voltages to a second order polynomial gave an excellent adjusted correlation of 0.99 due to a fairly constant plasma volume.

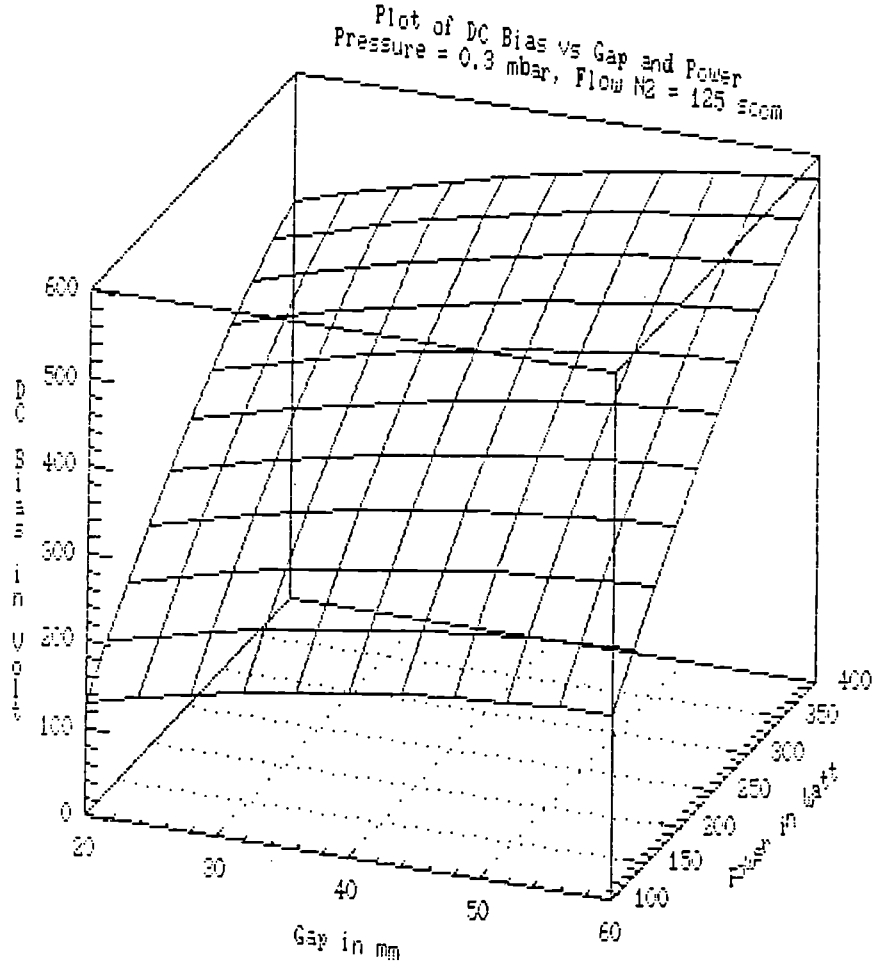


FIGURE 5
Performance of reactor MPE 3003. Plot of DC bias vs. gap and power at constant nitrogen pressure of 0.3 mbar and flow 125 sccm.

5. CONCLUSIONS

The three chamber reactive ion etching system MPE 3003 for 200 mm sub-micrometre pattern transfer in Al (Si, Cu) has been described. Preliminary results on RIE etch homogeneity and other important etch criteria such as bias voltage as a function of electrode gap, pressure and applied powwer are given. These have shown an excellent correlation between predicted and observed values. Results on aluminium etch modelling regarding kinetics and chemistry have also been given. Pattern delineation for 0.4µm feature sizes by X-ray lithography with SiO_2 is also reported. This has illustrated an etch homogeneity of better than 98%.

As part of the continuing ESPRIT programme 574, this machine will be used for further process development of the Al(Cu, Si) system, [11]. Extensive diagnostics and modelling of this multi-step etch process will also be performed.

ACKNOWLEDGEMENTS

We would like to acknowledge that this work was funded by CEC in part via ESPRIT 574 contract. Thanks to Peter Banks, Min Pang and Gerhard Lorenz for critical reading of the manuscript.

REFERENCES

[1] I. Hussla, K. Enke, H. Grünwald, H. Stoll; J. Phys. D. Appl. Phys. 20 (1987).

[2] H. Mader, private communication.

[3] R. W. Dreyfus, J. M. Jasinski, R. E. Walkup, G. S. Selwyn,Pure and Appl. Chem., 57, 1265 (1985).

[4] R. A. Gottscho and T. A. Miller, Pure and Appl. Chem., 56,189 (1984).

[5] K. O. Park, Proc. 4th symp. on Plasma Processing, The Electrochemical Society Inc., Pennington, USA, Proc. Vol. 83 (10), 300 (1983).

[6] R. d'Agostino, F. Cramarossa, S. De Benedictis and F. Fracassi, Plasma Chem. Plasma Proc., 4, 163 (1984).

[7] K. O. Park and F. C. Rock, J. Electrochem. Soc., 131, 214 (1984).

[8] B. J. Curtis, Solid State Technol., 23 (4), 129 (1980).

[9] J. W. Coburn and M. Chen, J. Appl. Phys., 51, 3134 (1980).

[10] I. Hussla, H. Birck, W. Katzschner, A. Pawlakowitsch, P. Sommerkamp, R. Schleiff, W. Pilz, T. Sponholz and H.C. Scheer, Proceedings CIPG 87, 4th International Symposium on Dry Etching and Plasma Deposition in Microelectronics, Antibes, France, p.90, (1987).

[11] In preparation for Microcircuit Engineering 87, International Conference on microlithography, 1987 Paris, France.

Project No. 958

HIGH PERFORMANCE VLSI INTERCONNECTION SYSTEMS

Project Partners : BULL SA
BRITISH TELECOMMUNICATIONS PLC
GEC RESEARCH LTD., MARCONI RESEARCH CENTRE

Project Leader : Karel KURZWEIL, BULL

Authors : P. ARROWSMITH (BT), N. CHANDLER (MARCONI),
G. DEHAINE (BULL)

1. ABSTRACT

Project 958 is concerned with the development of two major interconnect technologies : High Density TAB and a compatible High Density Multi-level Substrate Technology. These technologies together with the mechanical, thermal and electrical design and test tools and connector technology, developed to apply them, will be demonstrated in the course of the project in a multi-chip, high power module (about 360W) containing Gbit/s logic functions.

The project goals for TAB are to demonstrate both bumped chip and bumped tape TAB at 100 microns pitch and with more than 200 leads. For the substrate, to demonstrate four layers of logic track with 125 microns pitch track capability, over a conventional board system for power distribution and capable of Gbit/s rates in strip line.

The progress of the project towards these objectives will be presented and this paper also gives the structure of the project and some background to the technologies being developed.

2. INTRODUCTION

Silicon integrated circuit technology continues to evolve ICs of higher complexity and higher speed and the pin out requirements per IC continues to rise as also does the power density. Currently available packaging and interconnect technologies noticeably lags the performance required to fully exploit these new IC technologies [1] and will limit their potential use in new systems.

New and innovative approaches to packaging and interconnection are needed to narrow the gap between the performance of the basic IC and the performance of the IC available in an assembled system. Towards this objective, project 958 is developing high performance chip wiring and substrate technologies together with the essential mechanical, thermal and electrical design and test tools to apply these technologies in an integrated manner to the assembly of new systems. An important work is also carried out in the area of modelling of the electrical, thermal and mechanical aspects of packaging. Though the individual technologies and design and test tools developed in the project will be applicable separately, an important part of this project is to demonstrate a fully integrated approach to interconnection and in the course of the project, a multichip high performance module will be constructed to evaluate the integrated use of these new technologies.

There are two major technology thrusts in the project :

a. the development of TAB (Tape Automated Bonding), both in bumped chip and bumped tape versions from the current minimum geometry of about 200 microns, in progressive steps to 100 microns pitch and to more than 200 leads.

b. the development of a compatible multi-layer substrate technology with four signal layers in addition to the component mounting layer and power distribution layers. The signal layers to be capable of useful operation with Gbit/s logic and capable of about 125 microns pitch tracks.

The design and testing tools to use each technology are being developed in parallel with the technology and in addition chip protection, thermal management and module connectors are being explored to provide an integrated interconnection technology for advanced modules.

In this paper, the organization and programming of the project as a whole is outlined first and then the current progress of the technologies is detailed. An appendix compares the technologies being developed in this project with 'conventional' and alternative approaches and explains why they were chosen as the preferred approach to high performance interconnect.

3. PROJECT 958 ORGANIZATION, TARGETS AND TIMESCALE

The common objective of the partners in project 958 is to develop the necessary interconnect technologies and demonstrate their application in a high performance multi-chip module. This module will be based on a multilayer substrate about 10 cm x 10 cm containing four signal carrying layers. It will contain about fourty TAB bonded ICs up to 1.2 cm x 1.2 cm with a total dissipation around 360W and it will have active ICs operating at Gbit/s rates.

Very few companies in the world have the capability to produce such high performance modules and the engineering to make these modules necessitates the development of very high density TAB to 125 microns pitch and with more than 200 lead capacity and a compatible high density multi-layer substrate technology also capable of 125 microns pitch and the design and test tools to integrate the use of these technologies.

3.1. Organization

BULL S.A. is the prime contractor and in addition to the overall management, is responsible for :

1. The definition and evaluation of the demonstration module.
2. Bumped chip TAB development and evaluation.

BRITISH TELECOM is responsible for :

1. Bumped tape TAB development and evaluation.
2. Provision of ICs for test structures and the demonstration module.

GEC RESEARCH is responsible for :

1. The high density substrate development.
2. Thermal management,
3. Mechanical IC protection.
4. Repair and modification considerations for modules.

3.2. Targets

The advances in interconnect technology, which are the aims of the project, are outlined below compared with leading available technology at the start of the project.

	Available	Target
TAB		
(Bumped chip and bumped tape)	40 - 100 leads 200 microns pitch	> 200 leads < 125 microns pitch
Substrate		
Minimum line width	150 - 200 microns	50 - 80 microns
Minimum via size (pth)	300 - 400 microns	50 - 100 microns
Module		
Power density	1 W/cm2	5 - 10 W/cm2
Silicon : substrate area ratio	about 0.13	about 0.3 - 0.5
Chip size	10 - 50 mm2	> 100 mm2

These targets are very aggressive and several phases in the project have been defined to establish intermediate targets for the technologies.

Phase 1 : To establish initial parameter targets, i.e. the substrate configuration (structure, materials, complexity), the chips to be used, the targets for lead pitch and assembly equipment.

Phase 2 : An engineering definition of the test module and definition of target specification and testing, also definition of intermediate test pieces.

Phase 3 : Fabrication of components (tapes, substrates and ICs), development of thermal modelling.

Phase 4 : Assembly of test structures.

Phase 5 : Assembly of the final demonstrator module and evaluation of its performance.

The project is currently in phases 3 and 4 and as reported in the next section, good progress is being maintained towards the final target specifications.

3.3. Timescale

The project is planned to be completed in three years and is currently at the half way point. However, because of the demanding technology necessary for the final demonstrator module, all of the technology elements are already scheduled for demonstration and this has been achieved.

4. TECHNOLOGY STATUS

4.1. TAB

Two TAB technologies are being developed in the project, bumped chip TAB by BULL and bumped tape TAB (BTAB) by BRITISH TELECOM.

Both technologies are under development because they are seen as complementary and hence it was necessary that the project was able to address both TAB technologies. Bumped chip requires additional wafer processing to add the bumps to the IC wafer and then achieves lead out bonding by a gold-tin alloy bond. Bumped tape requires no wafer processing but does need two photolithography steps for the tape. Lead out bonding is achieved by conventional gold-aluminium thermocompression bonding. The TAB technology best suited to a particular IC technology is being developed from the current geometry in steps to the target of 100 microns pitch.

The first step was to demonstrate 162 microns pitch and establish the equipment for ILB (Inner Lead Bonding i.e. the chip pads to the TAB frame) and OLB (Outer Lead Bonding i.e. the TAB frame to the substrate).

Photo 1 shows a test chip with 162 microns ILB pitch and Photo 2 the same chip mounted and OLB bonded again at 162 microns pitch. As a result of evaluation work on TAB design, it has been decided to keep the same pitch for ILB and OLB which gives the optimum strength and reliability for the TAB lead outs. However, this has placed severe demands on the targets for the substrate technology. Chip to substrate lengths are set at about 1 mm throughout this project.

The next milestone was to demonstrate 125 microns pitch TAB and this represents a major advance compared with existing technology and needed a re-evaluation of all of the TAB and bonding parameters.

125 microns Bumped Chip TAB

a. Chip Bumping

Straight wall gold bumping on the aluminium pads has been selected for the following reasons :

- This technology is well understood in BULL and is compatible with the Ti/W adhesion/barrier layer available and provides good pad sealing.

- A straight wall process is compatible with the bump sizes for 100 microns pitch TAB bonding.

b. Tape

Standard 35 mm tape formate has been chosen, because it will maintain compatibility with the tape handling equipment. However, this choice will reduce the spacing of the test pads to 320 - 400 microns and will require a new approach to attach the test card to the frame. 35 microns thick tin plated copper has been adopted for the lead frame. Tapes at 125 microns pitch were successfully realized as the part of this project.

c. TAB Bonding

ILB will use gold tin alloy bonding, but additional control of the ILB equipment has been introduced for the high lead out frames to avoid any possibility of silicon damage. This is achieved by controlling the parallelism of the bonding tool, implementing a low strength guage control and stroke control at the bonding head. OLB is to be by lead tin soft solder.

125 microns Results

Photo 3 shows a 125 microns pitch ILB TAB chip on the frame and photo 4 shows a detail of the OLB bonded chip at the same pitch.

A test substrate (10 x 10 cm with 36 chip bonding sites) has been assembled at BULL, using G30 material (polyimide on pcb). These substrates are being currently used to exercise the OLB tooling prior to delivery of the project test boards. Photo 5 shows part of this board populated with the TAB bonded chip each having 284 lead outs at 125 microns pitch ILB and OLB. Photo 6 shows one assembled chip.

Status

125 microns pitch TAB has been demonstrated and the programme is on schedule to produce 100 microns pitch TAB by the middle of year 3 of the project ready for use in the test module.

4.2. BTAB

Bumped tape TAB is a more recent development in TAB technology which was developed at BRITISH TELECOM and is expected to be available in production in early 1988 at 200 microns pitch. In project 958, it is being advanced to 100 microns pitch in parallel with the bumped chip programme. The equipment for BTAB use is very similar to that used for bumped chip TAB, indeed OLB is identical. The primary differences are in the manufacture of the tape and in the ILB bonding which is thermocompression gold to aluminium.

Progress

In the research unit at BRITISH TELECOM, a pilot line for BTAB has been established to produce 100 microns pitch tape and carry out ILB and OLB bonding. In addition, a design suite has been developed to facilitate TAB designs and this is now available and was used to design the first project test frame (photo 7). The key element in bumped tape TAB is the formation of a compliant gold bump on the lead frame of a controlled height and hardness

80 x 80 microns bumps are currently being produced (photo 8) and have been produced with the necessary control for surface morphology (photo 9), height and hardness. Photo 10 shows a 200 microns minimum geometry tape frame with 80 microns bumps and photo 11 this frame bonded to a IC which has required no additional processing to prepare it for TAB bonding.

50 x 50 microns bumps can now be imaged and it is expected that the project 125 microns tape design with these bumps will be available before the end of this year and that BTAB at 100 microns pitch will be available during year 3 of the project as scheduled.

4.3. TWO LAYER TAPE TECHNOLOGY

All of the above work at BULL and BRITISH TELECOM uses three layer tape, so called because it consists of the backing plastic (usually Kapton), an adhesive layer and the 35 microns copper layer. As the pitch of the tape leads is reduced, it is increasingly difficult to control the etching of the copper layer which for 100 microns pitch is approaching 1:1 aspect ratio.

BRITISH TELECOM as part of the TAB programme is to evaluate two layer tape processing. This will only be available to the project in year 3 and the current status is that processing equipment is on order to obtain copper plated beams instead of the etched beams used in the current processes. Two layer tape has a thin (about two microns) layer of copper sputtered onto the backing tape and the plating up process is considered to be the preferred method of generating fine pitch TAB in the future.

5. SUBSTRATE DEVELOPMENTS

5.1. Materials

A large number of copper clad low dielectric constant laminates are commercially available, but two organic materials are most commonly used :

- Polyimide and Poly tetra fluoro ethylene (PTFE).

Of these two, PTFE is most attractive, due to its low moisture absorption, very low dielectric constant and ability to be supplied with reinforcement to achieve good dimensional stability. Unfortunately, it is only recently becoming available in the very thin form necessary for this project. Polyimide laminate was adopted at the beginning of the work, but PTFE dielectric is now also being assessed.

5.2. Structure

The process adopted for the multi-level substrate is to use these copper clad low dielectric constant organic laminates to produce a semi rigid high speed multi-layer circuit, which will be subsequently bonded onto a reinforcing module containing power distribution wiring (Fig. A). The alternative approach of sequential lamination onto a rigid mandrel or subcarrier did not lend itself to blind hole laser drilling or to subsequent blind hold plating of such small vias. However, this form of structure inevitably places great dependence on maintaining dimensional stability of the organic laminates during manufacture. A "ring of pins" tooling system (photo 12) has been developed to achieve the necessary registration in the photolithography and lamination processes and initial tests have produced side to side registration better than two microns.

5.3. Vias

In the signal layers of the substrate, the target for the vias was that they would be contained within the minimum pitch tracking of 125 microns to avoid wasting space with via pads. Hence, their diameter would have to be less than 50 microns. Tests were carried out with both Laser drilled holes and reactive ion etched (RIE) holes. RIE drilling was considered because it should produce better hole profiles and would produce all vias simultaneously. However, initial tests with RIE, using sputtered aluminium as the mask, failed to define the holes due to metal residues impeding the etch process. Extended Laser tests solved an earlier delamination problem and produced tolerable yields of 50 microns diameter holes on a 200 microns pitch as shown in photo 13 with a close up view in photo 14.

5.4. Test Substrates

Both as an intermediate step to the demonstration module and to assist the TAB development, five test substrates were defined in the project to be made by GEC RESEARCH. Three of these test boards are single layer substrates intended to assess TAB bonding and chip encapsulants and they have also been used to refine photolithography, plating and etching processes for the narrow tracks.

From this work, compensations have been incorporated into the CAD artwork generator for the narrow tracks and it has now been shown that the target dimensions can be achieved with both 12 and 25 microns thick resists. Two of the substrates require the multi-layer process using the registration tooling described earlier. One is a passive board for electrical tests and the second (Artwork 3) is a major intermediate step towards the final module.

Artwork 3

10 cm x 10 cm multi-layer technology.
Wiring pitch 125 microns.
Bonding layout for :
36 x 1 cm square test chips, each chip with 284 leads
at 125 microns pitch, capable of 10 W dissipation.

TOWARDS THE FINAL DEMONSTRATION MODULE

To produce the final demonstration module, a multiplicity of hardware and design techniques must be brought together by the project, in addition to the TAB and the multi-layer substrate technology described previously. In this section, these activities are summarized and how they impact the definition of the demonstrator module.

6. THERMAL MANAGEMENT

The 100 cm2 final module is intended to operate at 360 W and hence represents a considerable challenge to the thermal engineers.

6.1. Air Cooling

Thermal modelling exercises have shown that thermal resistances of 3°C could be achieved with these power densities and air velocities and 5 m/s for a single device. However, for closely spaced arrays of such high power devices, even the provision of pre-cooled air to each chip individually is unlikely to constrain junction temperatures adequately [2].

6.2. Liquid Cooling

Further thermal modelling was carried out to establish the thermal resistance of published advanced water cooled solutions to this problem. The analysis indicated that a 36 chip array with each chip dissipating 10 W could sustain a chip temperature of 100°C. However, the performance of such modules is very dependent on mechanical tolerances and assembly quality. Each of these heat exchanger modules is very complex and has to be specifically designed for each board assembly.

6.3. Direct Immersion Cooling

Additional studies were carried out on direct immersion in fluorocarbon fluids chosen for their chemical inertness and stability [3]. This means that naked chips can function satisfactorily whilst totally immersed in the fluid. The study revealed that the heat from a complete array could be removed by the immersion of a module in a tank of static fluid whilst nucleate boiling takes place on the back surfaces of the chips. If the energy from each chip were to rise, then it is anticipated that the transition from nucleate to film boiling would occur in the region of 15 to 20 watts/sqcm. This would cause an insulating layer of vapour to cover the chip, resulting in a rapid rise of junction temperature [3]. The module operating temperature may be further

reduced by forced circulation of the fluid, and by cooling the liquid below its saturation temperature. Roughening the back surface of the chip to provide nucleation sites will also reduce temperature overshoot prior to onset of nucleate boiling.

6.4. Summary

For experimental purposes during the project, direct immersion cooling will be used. As shown in Figure B, it is far more efficient than even forced flow gas systems. Although this form of cooling has been used on commercial equipment, efforts will continue to be made to find a more orthodox solution which will avoid some of the fluid leakage and other difficulties associated with this form of cooling.

7. DIE ATTACHMENT AND PROTECTION

The choice of encapsulant for the flip TAB bonded chips is influenced by a number of factors :

a. There is a maximum allowable spacing of 60 microns between the face down chip and the substrate surface.

b. Because of the face down attitude of the chip, the encapsulation should act as die attach to provide mechanical constraint to the chip.

c. As the whole assembly is likely to be immersion cooled in fluorocarbon liquid, the encapsulant must be effective in this medium.

It is known that the immersion of some materials in fluorocarbons [2] can cause swelling and displacement of plasticizers. Also, the low surface tension of the fluids makes module sealing difficult [4]. For these reasons, a series of tests is being conducted to investigate the compatibility of the chosen encapsulants with the fluorocarbon fluids.

7.1. Failure Mode

No data is available on the long term reliability of TAB bonded integrated circuits in storage or immersed in a fluorocarbon fluid. The indications are that in the fluid, it should be greatly enhanced, due to the considerably reduced likelihood of the presence of water which forms the corrosion cell necessary to promote the most common chip failure mechanism. If the fluorocarbon excludes moisture from the chip surface, then no matter how many free chlorine ions are made available, no corrosion cell can form.

7.2. Environmental Testing

It would seem that Temperature/Humidity/Bias testing would not be appropriate in this case, as the purpose of the accelerated life testing is to cause rapid failure of a system by the failure mechanism which would eventually occur in the normal operating life of the system. Thus the failure mechanism must be known before an accelerated test can be specified.

7.3. Current Activity

Several potential encapsulant/die attach materials are being tested for chemical compatibility with the fluorocarbon fluids at elevated temperatures. Further investigations are under way to identify appropriate environmental tests.

8. CONNECTORS

Due to the high signal speeds required, impedance discontinuities in the lines must be minimized. This means that special impedance matching connectors are almost essential. A number of possible systems have been identified, but the close spaced module I/O of approximately 1 mm is difficult to achieve. One possibility is to use special impedance matched proprietary flexible circuits with soft contact to mother boards.

9. ELECTRICAL ASPECTS

9.1. Test Chips

The module will be populated with both passive chips, to enable controlled heat dissipation patterns to be produced, and with a set of silicon ECL ICs to demonstrate that the substrate is fully functional to Gbit/s signals. The passive chips have been made at BRITISH TELECOM (photo 15) and the proposed ECL circuit (Fig. C) has been bench tested and BTAB frames have been designed for the ECL ICs and are currently in processing.

9.2. Strip Line in the Multi-Layer Substrate

The project target is the ability to accommodate 200 pico-second pulse edges. This requires that the signal wiring should have broadband capability into the GHz range. To accommodate this requirement, the wiring must be designed using "Transmission Line" theories, which involve the use of signal ground planes and close control of signal line impedances. To accommodate the wiring pitch requirement of 125 microns with printed circuit manufacturing technology, a line width of 65 microns was selected, together with gap width of 60 microns, to provide the target line impedance of 75 ohms, in covered microstrip format, for a dielectric constant of 3.5. The total laminate thickness is 190 microns from the top layer, which carries the device attachment pads, to the ground plane, and the two signal planes carry orthogonal tracks, to reduce crosstalk.

The required thicknesses for 75 ohms of the copper conductor and the surrounding insulator are within the range of the multi-layer technology. Additional signal wiring can be accommodated in the two lower signal planes which are made with the same multi-layer technology. It is known that when transmission lines are placed as close together as is proposed above, crosstalk may become unacceptable. This will be assessed with one of the phase 1 test substrates and limits will need to be defined.

The test circuit in the module will allow "bit error rate" testing to assess the substrate performance with a fully operating digital circuit.

10. SUMMARY

All the hardware techniques and design aspects necessary to specify and make the demonstration module have now been addressed and the project is on target to produce this advanced demonstration. Substantial progress at the demonstrator dimensions has already been practically demonstrated in the two key technologies, TAB and multi-layer substrates. The test structures will show the integration of the mechanical, electrical and thermal technologies which will be used to produce the final demonstrator and which will clear the way toward applications planned in real world high performance electronic systems.

11. REFERENCES

[1] Sage, M., Packaging and Interconnection for High Performance Electronic Systems, BPA, London, 1986.
[2] Timko, N.J., Air-Jet Impingement Keeps Computer Cool, Electronic Packaging and Production, May, 1987.
[3] Danielson, R.D., Thermal Management of Electronics by Liquid Immersion, Industrial Chemical Products Division, 3M Company.
[4] Danielson, R.D., Cooling a Superfast Computer, Electronic Packaging and Production, June 1986.

12. KEY WORDS

TAB, BTAB, CHIP BONDING, TAPE, BUMPED TAPE, COOLING, THERMAL MANAGEMENT, CHIP PROTECTION, PACKAGING, INTERCONNECTION, ILB, OLB, MULTI-LAYER SUBSTRATE,

13. TABLE A1

CHIP WIRING	WIRE BONDING	FLIP BUMPED CHIP	TAB
Current pad pitch/pad size	200 um/100 um sq.	200 um/100 um sq.	200 um/100 um sq.
Potential for reduction	poor	good	good
Re-Assemby Test	fair	fair	very good
Robustness	fair	excellent	good
Maturity	excellent	good	fair
Availability	excellent	fair	poor
Thermal Shock	good	?	good
Area Bonding	poor	very good	fair

APPENDIX

The objective of project 958 is to assemble the technologies to be able to make high performance multi-chip modules, which can fully exploit the performance of advanced VLSI ICs. Many aspects of the technologies chosen for the project differ from current conventional practices, but it may be of value to explain some of the reasons why three of these were adopted.

TAB for chip wiring, ommitting chip packaging and a very fine geometry multi-level substrate.

Why TAB for Chip Wiring ?

The table A1 summarises some aspects of the available chip wiring technologies. Wire bonding is rejected because of its poorer development prospects. Flip bumped chip has good potential for scaling, but there remain doubts of developing a sufficiently reliable system against thermal shock, particularly if the bump size is reduced. TAB, while currently poor in availability, appeared the best development prospect and since this decision, several projections have also suggested that TAB is likely to become a major wiring technology.

Why No Packages for Individual ICs ?

On the TAB tape, the chip is handleable and testable, hence these reasons for putting each IC in a package are eliminated. The module package will provide mechanical protection. The absence of all the individual packages enables considerable improvements in size and performance, particularly speed.

Why Such a Fine Geometry Substrate ?

The target for the substrate technology was set to be compatible with the TAB technology and to try to narrow the gap between on chip interconnect and current board interconnect capability. On the IC, the minimum track pitch is six microns (three microns)*, vias are two square (one micron square) and there are two layers (four); equivalent to a density of 34 m/cm2 (123 m/cm2). On a typical current pcb, the minimum track pitch is 300 microns, vias are 0.2-0.3 mm and equivalent density excluding the degradation of vias is 50 x smaller (200 x smaller). The substrate technology targeted by the project, while not closing this gap, reduces the difference in density from 50 x per layer to 20 x per layer and does not lose any density in via placement, which is less than the track width. This has set a severe target for the project, but offers substantial improvements in system integration.

* ( ) figures are projected for one micron processes.

ILLUSTRATIONS

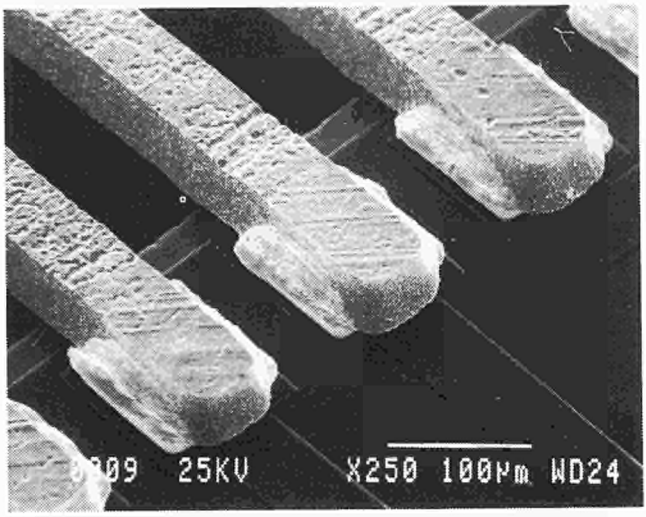


PHOTO 1
Inner Lead Bonding (ILB) at 162 Microns Pitch

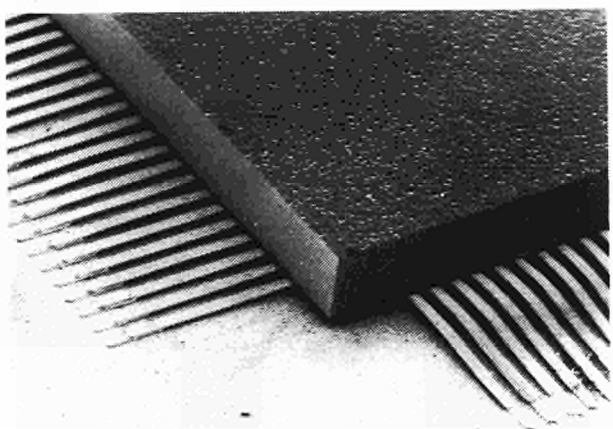


PHOTO 2
Detail of Outer Lead Bonding (OLB)

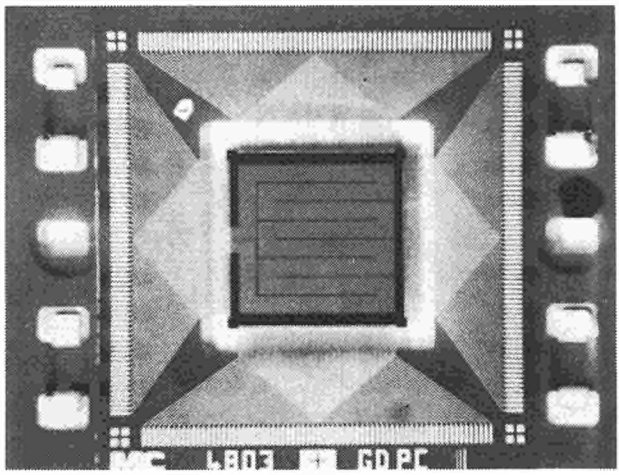


PHOTO 3
Test Chip (125 Microns) Pitch Mounted on 35 Microns Tape

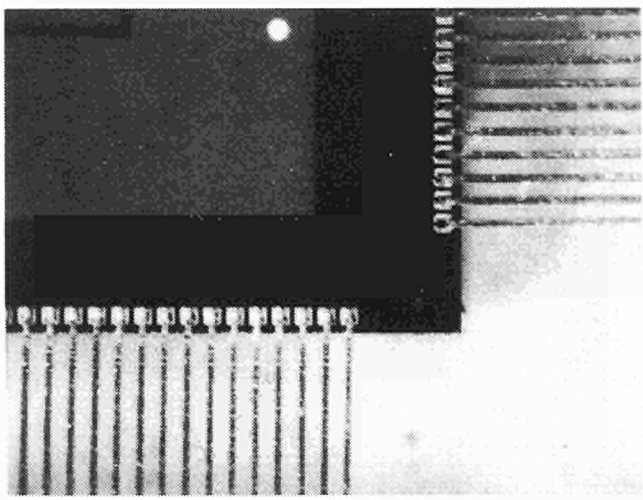


PHOTO 4
Detail of ILB Bonds at 125 Microns Pitch

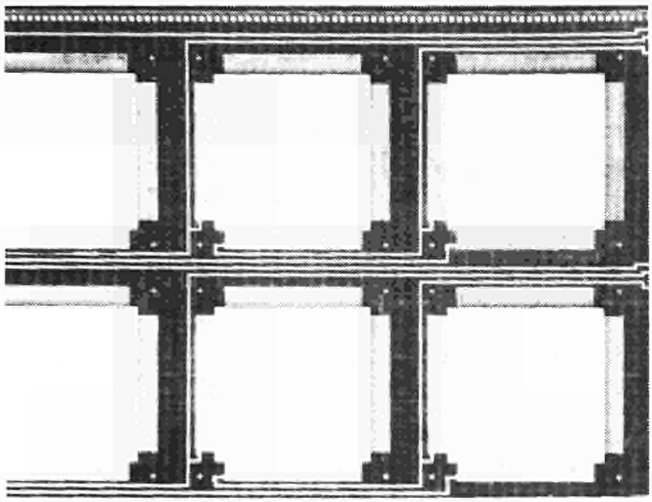


PHOTO 5
Partial View of Populated Test Substrate (125 Microns Pitch)

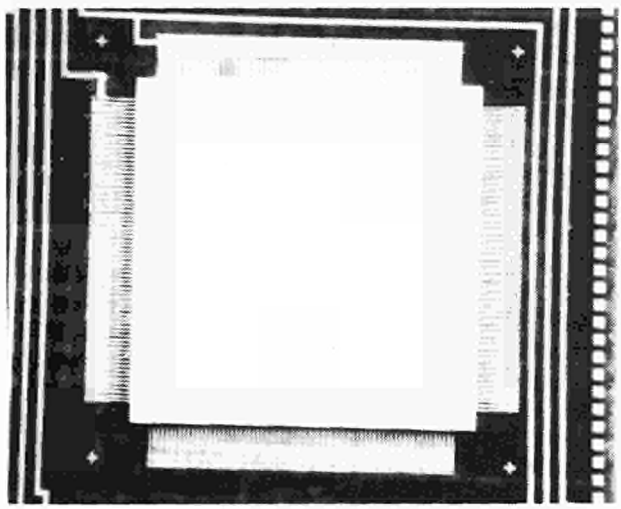


PHOTO 6
Close-up View of Chip Bonded on Test Substrate

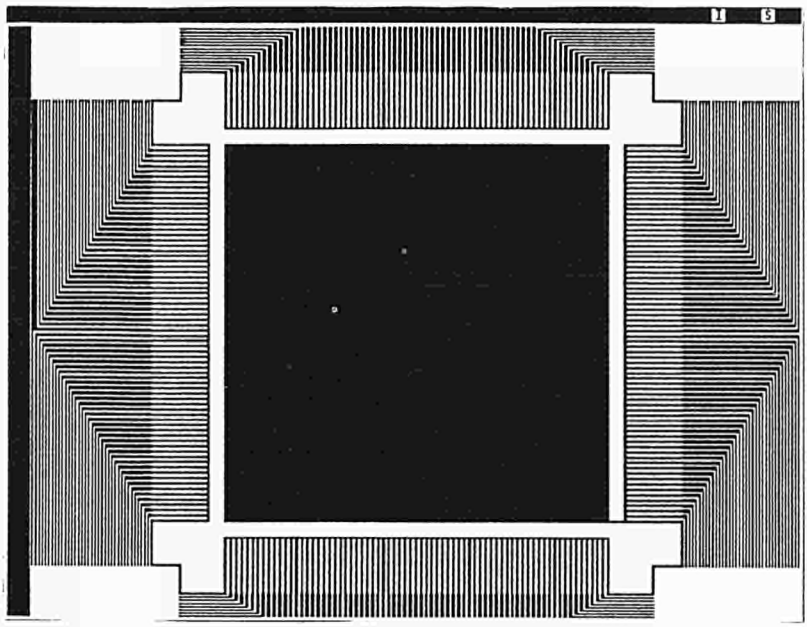


PHOTO 7
125 Microns Test Mask

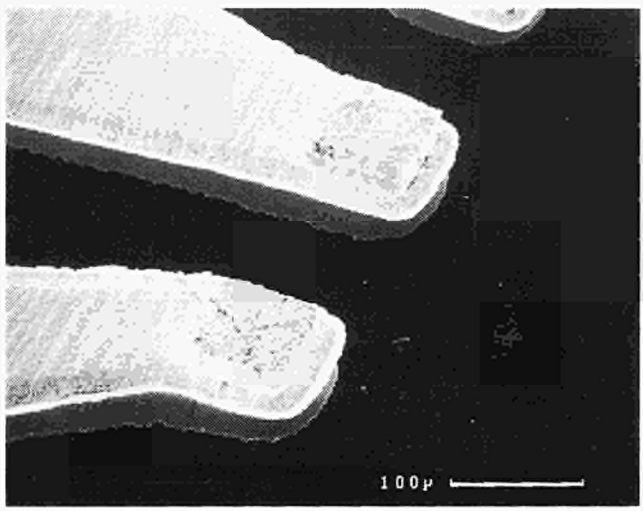


PHOTO 8
80 x 80 BTAB

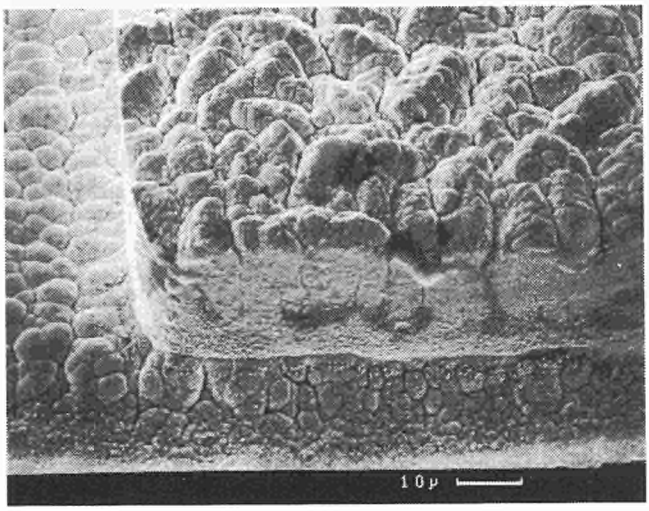


PHOTO 9
BTAB Close-Up

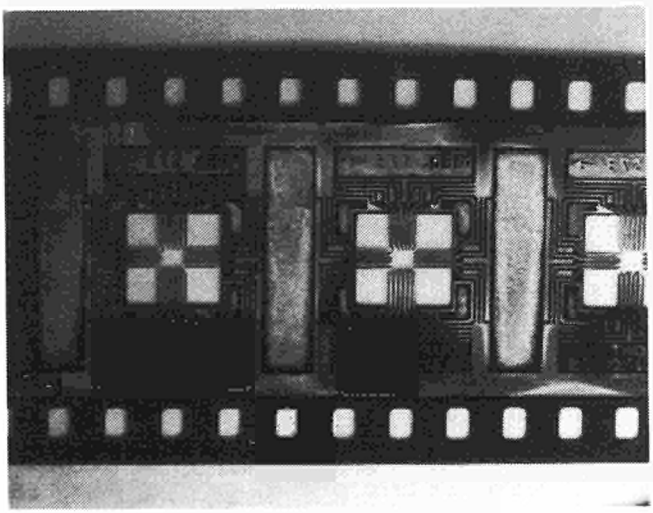


PHOTO 10
Strip BTAB

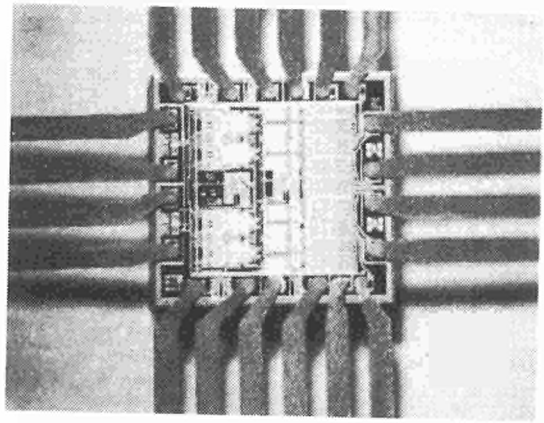


PHOTO 11
Bonded IC BTAB

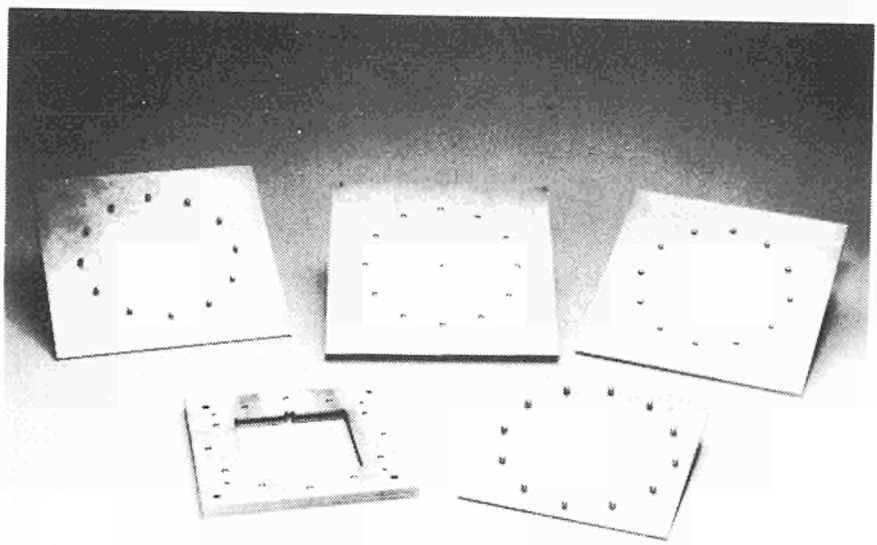


PHOTO 12
'Ring of Pins'

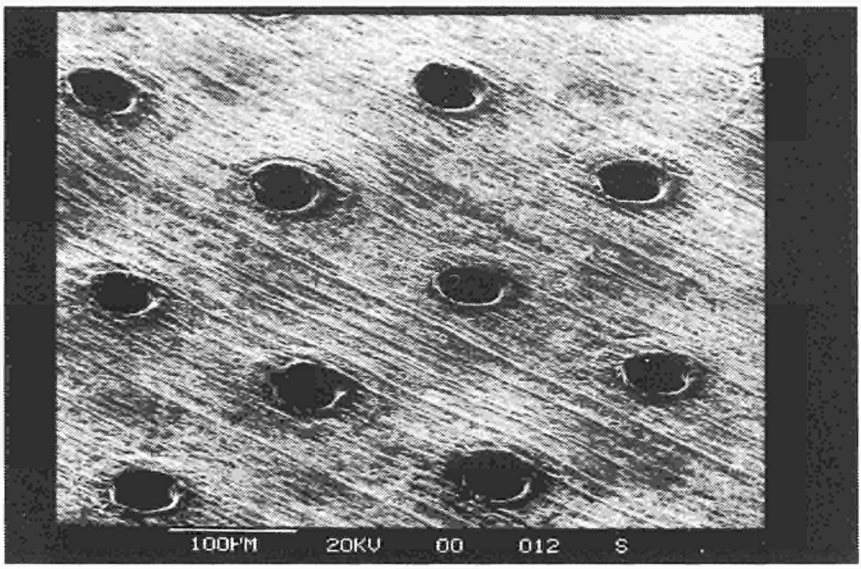


PHOTO 13
50 Microns Vias

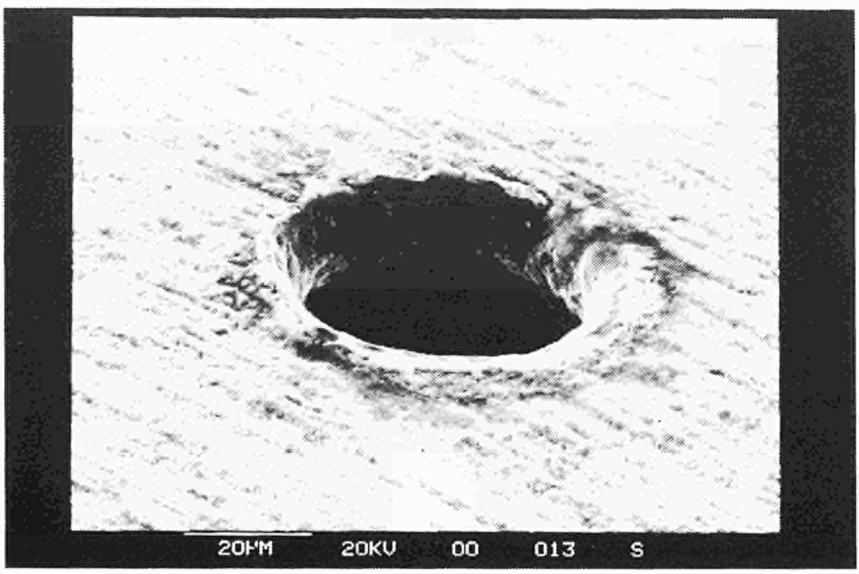


PHOTO 14
Close-Up of Via

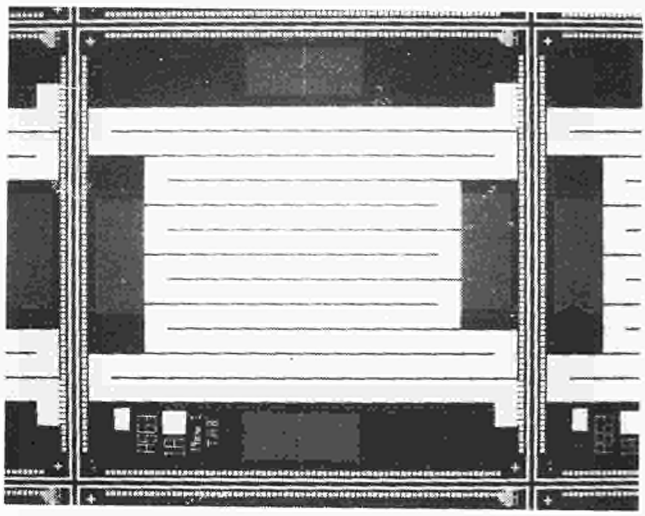


PHOTO 15
Passive Chip

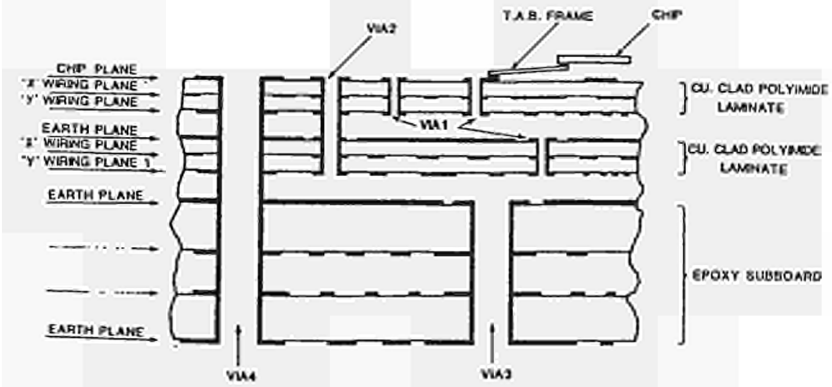


FIGURE A
SCHEMATIC CONSTRUCTION OF PROPOSED
PHASE 2 SUBSTRATE

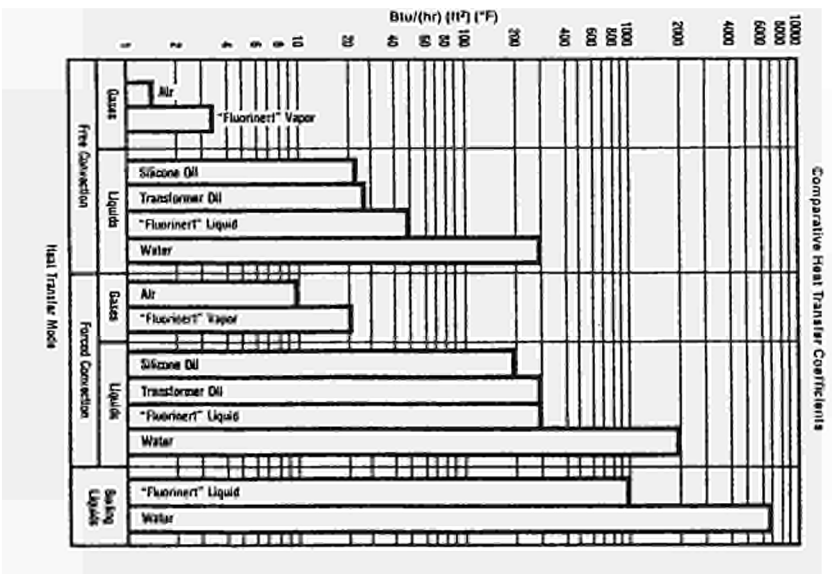


FIGURE B
EFFICIENCY OF VARIOUS COOLING METHODS

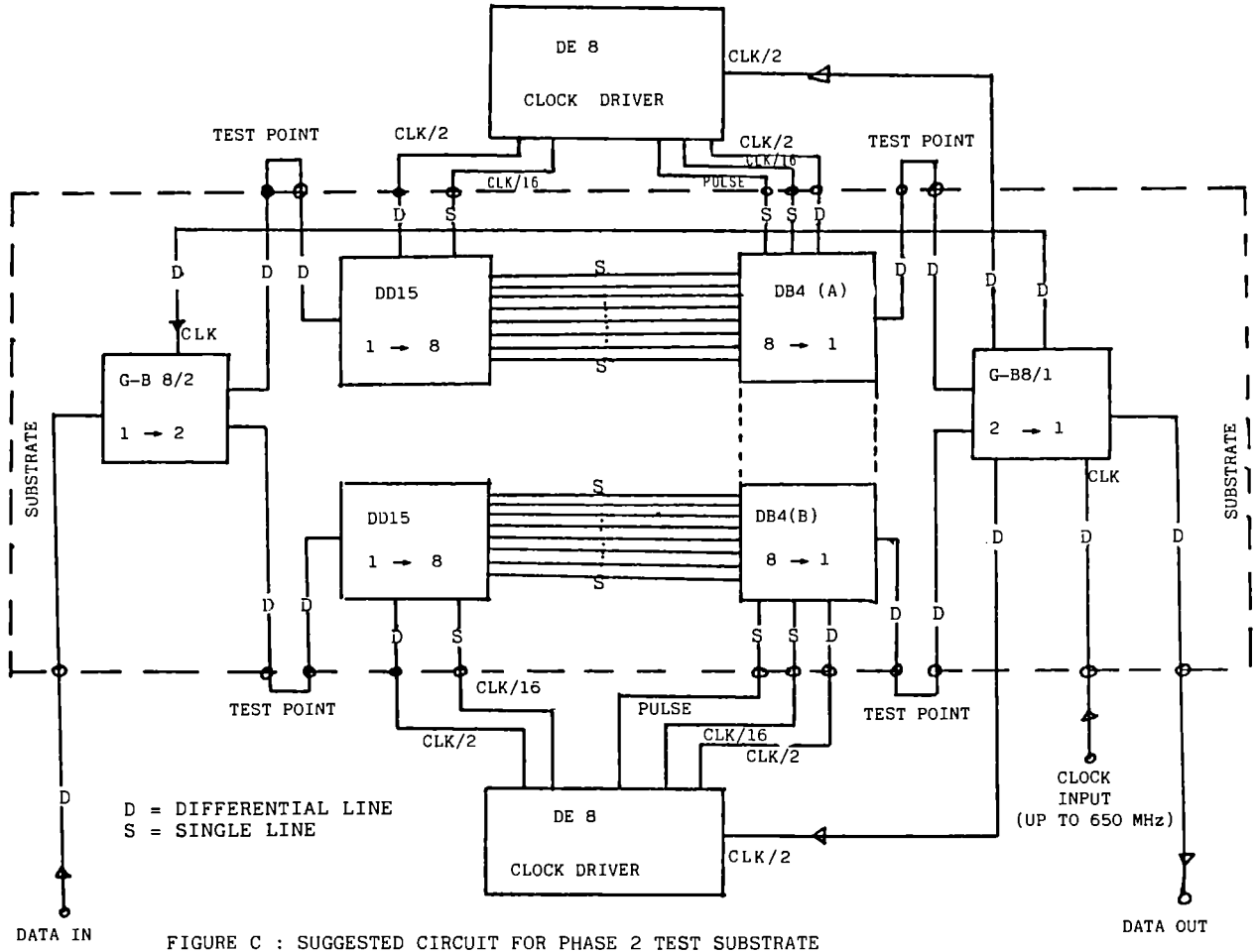


FIGURE C : SUGGESTED CIRCUIT FOR PHASE 2 TEST SUBSTRATE

Project No. 971

TECHNOLOGY FOR GaAs-GaAlAs HETEROJUNCTION BIPOLAR INTEGRATED CIRCUITS

THE ESPRIT 971 PROJECT TEAM : M. Bon and al. (CNET, Bagneux, Fr.), A. Rezazadeh and al. (GEC, Wembley, U.K.), R. Goodfellow and al. (PLESSEY, Caswell, UK), W.M. Kelly and al.(FARRAN TECH., Cork, IRL), D. Carr and al. (PLASMA TECH., Bristol, UK.)

ABSTRACT :

The objective of ESPRIT Projet 971 is to develop high performance GaAs/GaAlAs process for ultra-fast HBT ECL logic. The three main european companies involved in HBT ICs, GEC, Plessey and CNET, are cooperating in that context with the additionnal support of Farran Technology and Plasma Technology as subcontractors. During the first year of the project, major results have already been obtained in epitaxial growth (both MBE and MOCVD), basic and advanced self-aligned technology and modelling. The basic process, although using still conservative design rules of 2-3 μm, has already allowed fabrication of divider-by-four operating above 5 GHz using HBTs with cut-off frequencies above 15 GHz, in close agreement with simulations and giving thus high confidence in the expected performances of the self-aligned micron and submicron processes to be assembled in the future.

1. INTRODUCTION

The aim of Project 971, which started in February 1986, is to develop a high performance GaAs process using GaAs-GaAlAs HBTs for ultrafast emitter coupled logic. With the help of Farran Technology for e-beam lithography and Plasma Technology for plasma deposition and etching, the three main partners CNET, GEC and PLESSEY are investigating all aspects of the IC process including material epitaxy (both MBE and MOCVD), ion implantation, rapid thermal annealing, lithography, dry etching, ohmic contacts including refractory alloys, and dielectrics. This work is accompanied by a strong effort on modelling, concerning both 1-D and 2-D device numerical modelling and analytical C.A.D block models. Process validation is established by fabrication of SSI/MSI (ECL) circuits such as dividers, multiplexer or arithmetic ICs.

The first two years of the program are dedicated to the development of a basic process validated on SSI divider circuits and to the investigation of an advanced self-aligned process. The second two years phase will be concerned with MSI and SSI demonstrators using those respective processes and with subsequent refinement of those processes.

This paper will highlight the most significant achievements obtained during first year of the project and at the beginning of year two. These include a high degree of quality and uniformity of both MBE and MOCVD layers, world record values for p-type base doping, assembling, validation and comparison of three basic technological processes using ion-implantation, rapid thermal annealing (RTA), dry etching and high performance n-type and p-type ohmic contacts (including new original AuMn alloy). Also several approaches leading to self-alignment and micron and submicron rules have been investigated with already a very exciting successful fabrication of exploratory self-aligned HBTs. On the other hand all basic processes have been characterized on HBTs

with about 15 GHz cut-off frequencies, and dividers-by-two have been demonstrated with input rates as high as 5,7 Gbits/sec.

In the following, the 971 Project achievements are reviewed in the successive areas of epitaxial growth, basic and advanced technology, test and modelling, and finally HBTs and ECL ICs demonstrators.

2. EPITAXIAL GROWTH

Epitaxial growth quality control is a crucial point for HBT manufacturing and major improvements have been obtained in that area by Plessey [1] and CNET with MOCVD, and by CNET and GEC with MBE, by optimizing growth procedures and sometimes the growth equipment itself. In these four cases, all three partners are routinely working with 2" wafers and have developped strong characterization actions to support this growth quality objective. This involves regular use of SIMS and CV profiling, RX, PL, AES, Hall, TEM, SEM, optical inspection and some other occasionnal techniques for detailed wafer and layer assessment and optimization. A specific collaborative action for detailed cross-evaluation of SIMS, Hall and CV profiling techniques used by the different partners is being done with epilayers circulating between partners for this matter. Thickness uniformities have been improved in all four cases to values better than +/- 5 % on 2", best values being obtained by Plessey for MOCVD (+/-3 %) and by CNET for MBE (+/- 1 %). Best results for n and p doping and emitter aluminium content are also now a few percent with adequate abruptness control in all cases. Morphology is also a major concern and both MBE teams now routinely get oval defect densities of a few 100 cm^{-2}.

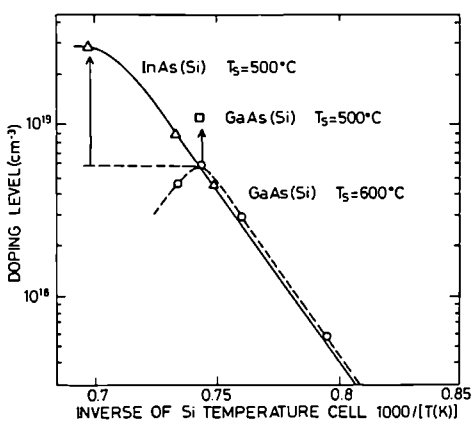


Fig. 1 : Maximum n-type doping of MBE InAs compared to GaAs.

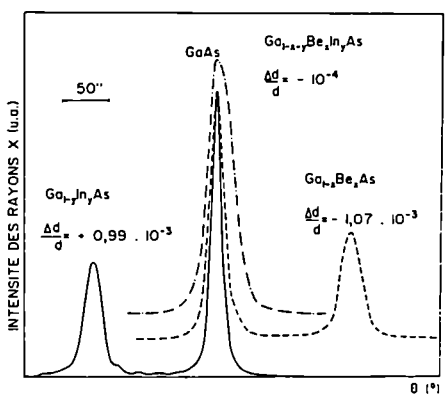


Fig. 2 : X-ray spectra of highly doped MBE GaAs:Be, GaAs:In and GaAs:BeIn.

All partners use similar standard structures for regular processing. These structures are already rather complex since they include five basic layers ($E(n^+)$, $E(N)$, $B(p^+)$, $C(n)$, $C(n^+)$), generally separated by graded layers. Nevertheless much effort is still done to push forward the HBT performances by introducing more sophisticated layer structures. For instance GEC has made an optimization study of the E-B interface layer and demonstrated higher DC current gain by using a specific chirped superlattice.

Many other techniques of the "bandgap engineering" type are also currently looked at, especially by partners using MBE growth (CNET and GEC). They involved E-B spacer, superlattices or quantum wells for different layers (E, E-B, B, C-B, C-S), and bandgap grading for the base as well as for the contact emitter layer. In this last case CNET has demonstrated the possibility to get extremely low specific contact resistance (10^{-7} Ohm.cm^{-2}), without alloying anneal, using a graded highly doped (3.10^{19} cm^{-3}) GaInAs contact layer (fig. 1) and comparison with a parallel work by Plessey using MOCVD should be made in the near future.

But one of the major point of concern, with respect to these non-standard structures, is the search for high and ultra-high p-type doping for the base in order to minimize the base resistance under the constraint of not degrading the transit time unacceptably. In that area, CNET has obtained world-record doping levels up to 4.10^{20} cm^{-3} using Be dopant in a MBE Riber 2300 machine. The small lattice mismatch (10^{-3}) generated by this ultra-high doping level has been compensated by isoelectronic In incorporation (fig. 2). Complementary work is under progress to establish the best compromise between base doping level and thickness and wafer processing, but already HBTs with gain value of 35 and low base sheet resistance of 150 Ohm.square have been demonstrated.

3. TECHNOLOGY

During the first year of the 971 project, the partners developped a basic process, compatible with 2-3 um design rules, validated by HBTs and dividers ICs (see 5), and explored new technological steps for a self-aligned advanced process.

3.1. Basic technology

All three partners have developped a basic process using emitter-up configurations grown on 2" S.I. substrates (fig. 3). However different technological options for the different process steps are developped by the different partners and systematically compared between themselves.

For the p-type implantation to contact the base layer, CNET has compared Zn and Mg implant, Plessey has focussed on Mg and GEC on Be. Both rapid thermal annealing (RTA) in optical furnace and classical furnace annealing (CTA) were initially compared. As a result, all partners agreed to select RTA as best choice with respect to various criteria such as limited unwanted diffusion (in-, out- and lateral), surface concentration, profile control and activation. As for the dopant, both Mg and Be were selected using these same criteria. In the case of Mg, a detailed cross-analysis of RTA temperature diagram, activation level and doping profiles was thus possible and effectively done, with samples circulating between partners. It is to be noted that potential advantages of Be over Mg in term of diffusivity and perhaps solubility remain to be confirmed and will have to be significant enough to compensate the hazard problem which makes many laboratories reluctant to its use.

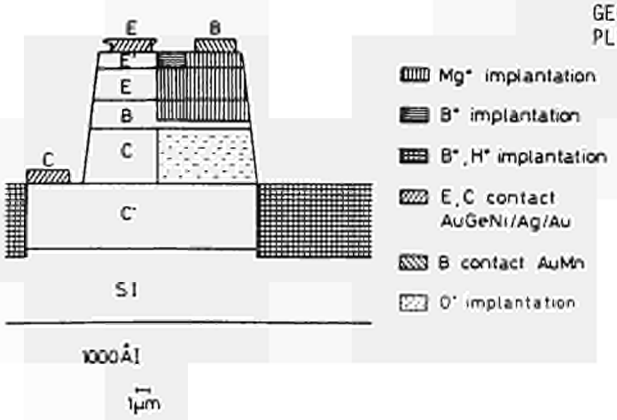
This process step is closely linked to the p-type contact metallization for which are compared classical AuZn (Plessey, GEC) and newly developped

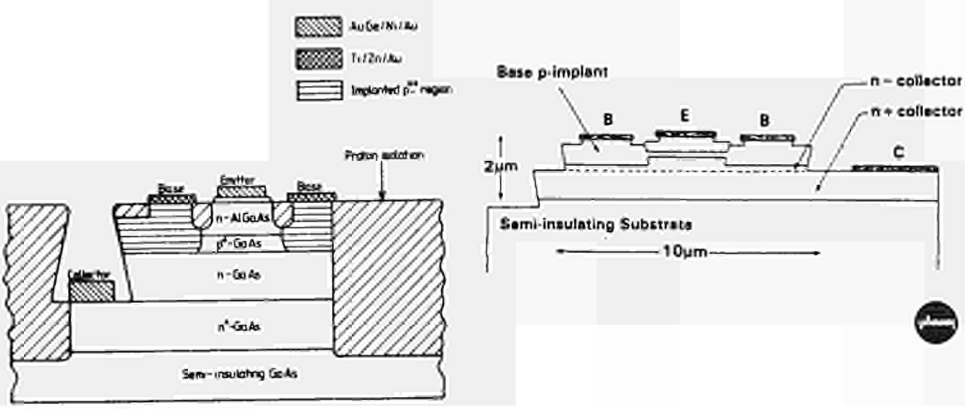
AuMn (CNET). Although the comparative characterization is not finished, current results show a much lower ρ_C for AuMn with record values as low as 2.10^{-7} Ohm.cm^2 on epitaxial layers, but with still some reproducibility problem on implanted layers which are currently being adressed.

N-type ohmic contact was also the subject of a comparative collaboration. InGeAu was selected by Plessey as giving better morphology and nearly equivalent ρ_C ($2.5\ 10^{-6}$ Ohm.cm^2) as NiGeAu when annealed by RTA. On the other hand, by pushing the optimization of both RTA and CTA for AuGeNi/Ag/Au, CNET has demonstrated in both cases record values of 2.10^{-7} Ohm.cm^2, with a small edge in favour of RTA because of better morphology but not so good as with InGeAu.

More process steps are currently the purpose of collaborative comparison and/or information exchange. They includes : H and B implantation for isolation, SiN and SiO_2 by PECVD, polyimide, resistors fabrication, dry etching (RIE and IBE) and substrate evaluation.

Fig. 3 : HBT structures processed by :
CNET (left)
GEC (left below)
PLESSEY (right below)





3.2. Advanced self-aligned technology

In that area, the purpose of the Project is to develop a self-aligned implanted process with micron, and subsequently submicron, design rules for digital ECL ICs working at speeds greater than 10 Gbits/sec. Two main actions are undertaken. The first one concerns e-beam lithography and associates CNET, GEC and FTL. The second one concerns the emitter contact cap layer and refractory metallisation compatible with the base implant annealing. Comparative work is done on these two subjects. CNET is looking at graded epitaxial InGaAs layers whereas Plessey and Plasma Technology (P.T.) are collaborating on PECVD Ge layers. GeMo/W refractory n-type metallisation were investigated during this first year by CNET and GEC and will be compared in the near future with W and WSix with collaboration of Plessey and P.T. which has already initialized research work on this subject.

3.2.a. E-beam lithography

The main objective here is to develop resist processes adapted to the various technological steps of HBT ICs manufacturing. This includes different kinds of lift off, chemical etching, dry etching, and implantation steps. Special attention is devoted to the throughput, which is a well known limitation of e-beam lithography, and also to the compatibility with the classical UV photolithography in order to obtain a flexible mixed e-beam/UV lithographic process.

The participation of GEC was to establish a reference starting point by applying current knowledge, using classical PMMA resist, to a simplified five levels HBT process on conducting substrate. This was done with the same EBMF 6.5 e-beam machine as the CNET one and with procedures similar to those common at CNET.

FTL rôle was to evaluate potentially useful commercial resists as well as newly sampled resists looking at the best compromises between resolution, sensitivity and process compatibility. A very complete survey has thus been done in close interaction with CNET for criteria specification and results validation. Using the guidelines resulting from this work and its own know-how on mono-and multilayered resist processes, CNET is currently developping a mixed e-beam/UV lithographic process for small design rules HBT ICs. The aim is to exploit e-beam both for its high-resolution and re-design flexibility, with a throughput much improved with respect to standard PMMA processes.

The above-mentionned compromises were found to be difficult due to insufficient capabilities of commercially available fast resists. CNET and FTL has thus evaluated multilayer resist systems which, among other avantages allow independent and complementary choice of base and definition layer, thus maximizing the effectiveness of the process in spite of some added complexity. This is especially the case for ion implantation and ion beam milling masking where a thick mask is necessary. For that matter CNET has investigated a trilevel system with an AZ UV-compatible and medium e-beam sensitivity definition resist, while FTL obtained good results using the fast e-beam resist FBM-120 (fig. 4) thus preparing an alternate higher throughput system to be transferred at CNET.

3.2.b. Emitter cap layers and refractory ohmic contacts for self-aligned HBTs

A strongly collaborative action has been undertaken by Plessey and subcontractor P.T. concerning P and As highly doped PECVD Ge layers using GeH_4 with PH_3 or AsH_3 gases. Good morphology and purity have been obtained by P.T on both phosphine doped and undoped Ge layers which have then been characterized

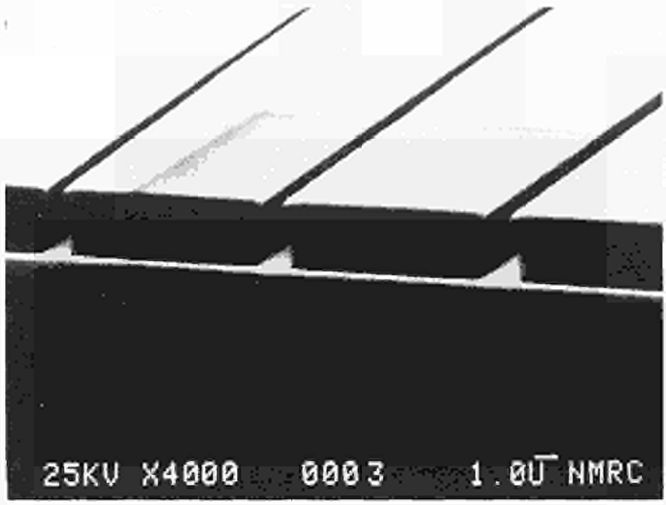


Fig. 4 : Three-level resist mask AZ1375/Al/FBM120

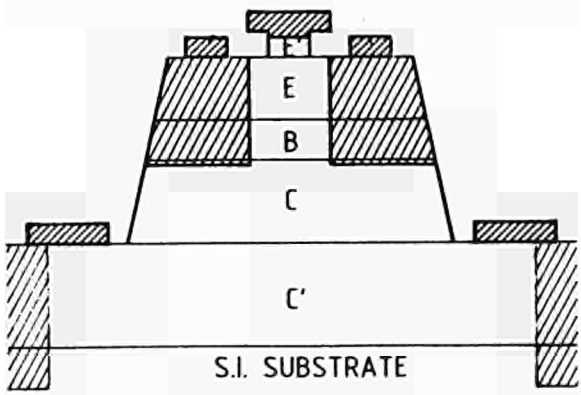


Fig. 5 : Self-aligned HBT test structure.

extensively by Plessey. This pioneering work has led to a low ρ_c of 2 10^{-6} Ohm.cm^2. The idea is here to use a low gap very highly doped layer for improvement of the emitter contact in a way compatible with a self-aligned process.

The same objective is followed by parallel CNET work on MBE grown graded InGaAs layers for which growth conditions have been established already leading to 3 10^{19} cm^{-3} Si-doping level (fig. 1) and to a record ρ_c value of 10^{-7} Ohm.cm^2, which is close to the minimum measurable value by TLM method.

Concerning refractory ohmic contacts, different variants of the GeMoW approach have been investigated using sputtering (GEC) and mixed sputtering/e-beam gun (CNET). CNET has compared RTA and CTA under As overpressure and obtained very good ρ_c values of 3.10^{-6} Ohm.cm^2 in the last case. GEC has looked at a more convenient technique, using a Si_3N_4 cap layer during CTA, with rather good

ρ_c values of less than 10^{-5} Ohm.cm^2. More work is still needed to select an optimum procedure in this area. On the other hand, P.T. has initiated work on PECVD tungsten and tungsten Silicides.

CNET has developped a RIE process to selectively etch the GeMoW contact using a Pt mask and a multistep RIE sequence to obtain a T shape structure where the undercut determines the distance between the ohmic emitter contact and the p-type implantation (fig. 5). Here again more work is still needed, especially to eliminate the formation of residues and also to assess this technique against the use of spacers.

Nevertheless a first self aligned large dimension test transistor (SA-HBT) has been successfully realized by CNET and work is currently running on small dimension SA-HBTs using this GeMoW approach [2].

4. TEST AND MODELLING

In that area, this first year of activity has been largely devoted to development and validation of both hardware and software tools. DC and RF testing are now well established at all three partners locations with specific effort done by CNET concerning fully automated DC testing as well as deembedding and equivalent scheme extraction procedure from Sij parameters measurements, by GEC an Plessey concerning packaging and associated RF testing. All partners are also now using direct microwave measurement on wafers for HBTs, with the availability at Plessey for testing logic circuits up to 5 Gbits/sec on-wafer.

One-dimensional drift-diffusion models were already operating and have been refined by GEC (BIPOLE) and CNET (ETHER) while 2-D finite element simulation is nearly ready, including a friendly process-oriented input preprocessor developed by CNET (TITAN III-V). Both partners have confronted their approach and results in the area of physical simulation. These 1-D device physics-oriented simulators have been used extensively by CNET and GEC for detailed analysis and optimization of the HBT structure, concerning especially Aluminium composition and emitter-base grading, base width and doping, collector doping, various superlattices behaviour (cf. 2) and so on. In the case of GEC model for instance, both vertical an lateral analysis are carried out successively yielding the base and collector current as a function of base/emitter voltage and f_t as a function of collector current.

On the other hand all partners have also developped small and large signal models coupled with commercial ICs simulators (SPICE and ASTEC). For instance a distributed CAD-oriented transistor dynamic model has been developped by Plessey and validated by real devices measurement. From this small signal model, a simplified lumped element model has been derived which is used in SPICE simulator for complete ICs simulations with good coherence with measurement on ECL divider-by-four (see 5).

Although somewhat similar, the various structures and design rules used by the three partners present many differences and variations. In order to go further in the comparison between the different approaches using all models cited above, the partners have commonly agreed to exchange all necessary informations including a detailed quantitative description of epilayer structures, geometrical data and experimental electrical parameters data. This common disclosure of very sensitive data is a good measure of the level of cooperation and open information exchange that has been developped in the consortium.

5. DEMONSTRATORS

At this point of the project, demonstrators have been fabricated with the basic processes and both unitary HBTs and SSI ECL ICs are concerned. Note however than results on SAHBTs have also already been obtained (cf. 3.2.b).

All partners have fabricated, tested and compared HBT devices identical to the transistors used in their present ECL ICs design. In that case, design rules are still rather conservative with both emitter width and base implantation-emitter spacing between 2 and 4 μm. Nevertheless, the results obtained are a good indication of the very high performance potential of HBT technology. Indeed, with those conservative design rules using basic non self-aligned process, with many parameters still not optimized, f_t and f_{max} have been regularly GHz for f_{max} in the case of a 2 μm rules HBT of Plessey, a higher f_{max} of 17.8 GHz having been obtained with somewhat reduced geometries).

One of the short term objectives of the project was also to apply the basic HBT process to SSI ECL ICs in the 4 to 8 GHz range. Dividers by-two (fig. 6) and divider-by-four ICs were thus designed [3] and fabrication started by all three partners (fig. 7). This again allowed confrontation of approaches and simulation results during the design phase and is promoting still stronger exchange in the area of test and packaging.

Important results have already been obtained by Plessey who successfully fabricated divider-by-four (fig. 8 and 9) in two versions using respectively 4 μm and 2.5 μm emitter width and operating reproducibly well over 2 GHz and 3.5 GHz under direct on-wafer measurements. By individual adjustment of bias conditions, input rate values up to 5.7 GHz were obtained [4]. The circuit includes 44 transistors. It consists of two master-slave D-type latches in series and dissipates 700 mW typically. Effect of packaging has also been tested showing a 15 % decrease of speed for a conventionnal leadless carrier and a 10 % improvement for a 50 ohms microwave jig on a typical divider tested on-wafer at 3 GHz.

An interesting point which resulted from this confrontation of results between partners was that, when comparing CNET and Plessey results on HBTs, same order of magnitude of f_{max} and f_t were obtained although the design rules of Plessey were somewhat smaller that CNET ones. This showed the importance of optimizing passive parasitic parameters and especially the contact resistances on which CNET has focussed recently obtaining the results already mentionned in 3. In the case of Plessey design for example, simulation shows that the divider-by-two frequency could thus be pushed up to 11 GHz with the same layout.

6. CONCLUSION

At the present time, the 971 ESPRIT project has resulted in the development of a basic process for GaAlAs/GaAs HBT ECL ICs (2-3 μm rules ion-implanted non self-aligned process). The three partners involved, GEC, Plessey and CNET, have regularly compared their approaches in all areas and collaborated on specific actions which allowed to speed up their development pace and clarified the criteria and choices. A good level of performance and associated reproducibility has been obtained both in epitaxial growth (MBE an MOCVD) and process technology although of course much effort is still needed, especially on uniformity, reproducibility and yield aspects as well as on power dissipation in order to quickly push the ICs complexity toward the MSI level. This will be a major part of the work in the near future.

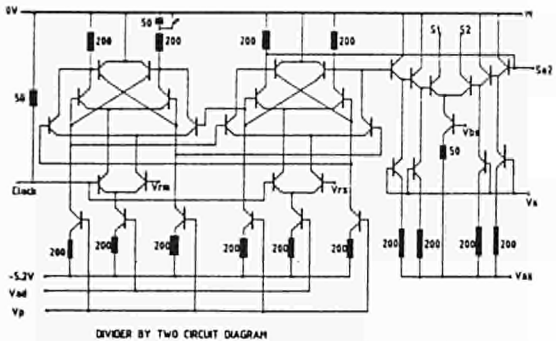


Fig. 6 : Master-Slave divide-by-two diagram

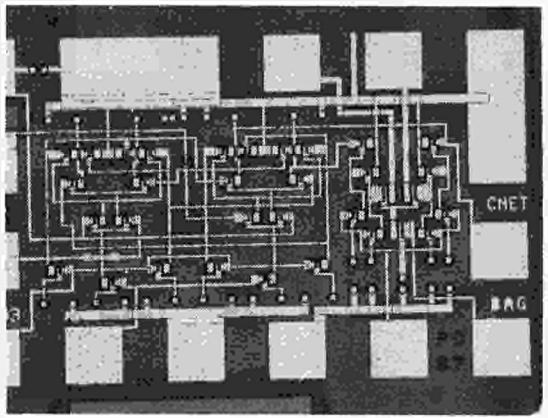


Fig. 7 : Micrograph of a divide-by-two IC

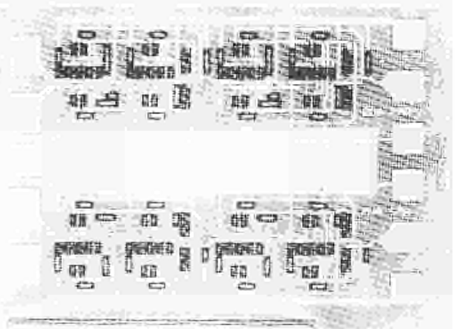


Fig. 8 : Micrograph of a divide-by-four IC

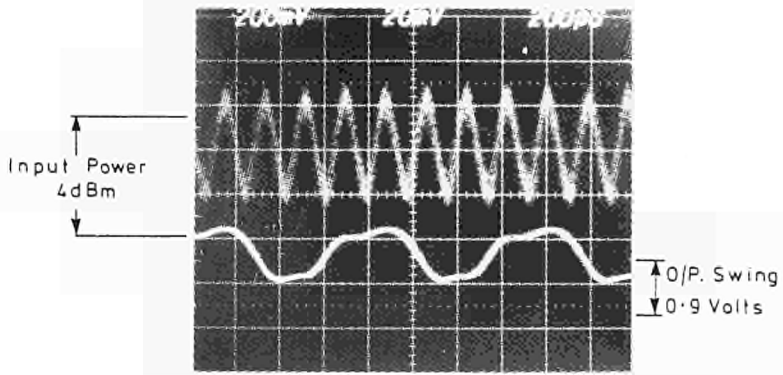


Fig. 9 : Divide-by-four IC tested at 5.4GHz on-wafer

On the other hand, one of the main teaching of this first year was the concrete validation of the fundamental advantages of HBT for ultra high speed logic applications. The demonstrators operate at already very high frequencies using the basic non self-aligned process in spite of the still conservative design rules which were chosen (2-3 μm) : maximum obtained performances being f_t = 16.6 GHz , f_{max} = 17.8 GHz and D/4 working at 5.7 GHz. This shows that such an unoptimized non self-aligned GaAs HBT process is nevertheless already on the same level of speed performance as an advanced super self-aligned Silicon Bipolar process [5]. Moreover the very good fit obtained between test and simulation data give confidence in the predictions established for the capabilities of the future advanced self-aligned process for which input bit rates above 10 GHz should be obtained using HBTs with f_t and f_{max} above 30 GHz. With such a self-aligned process Rockwell and NTT [6, 7] have recently obtained impressive f_t and f_{max} in the 50-70 GHz range (with 104 GHz announced) and dividers working at 13.7 GHz (with more than 20 GHz announced). This is a major challenge for our consortium for the near future.

The 971 ESPRIT project funded by EEC has allowed to promote detailed information exchanges, including very sensitive data, and cooperation focused on many points concerning growth, process, modelling, design and test of HBT ECL ICs. By stimulating this work and introducing cooperation under various forms between the three main involved European research laboratories and two additionnal subcontrators, EEC ESPRIT funding promotes the development of an European adequate response to the now buzzing activity in that domain in USA and Japan.

7. ACKNOWLEDGMENTS

A large number of people are participating in the 971 ESPRIT Project. Besides the teams' managers listed above as authors, the following people are to mention, due to their essential role in driving the activity of the different tasks of the differents partners, and in the coordination of collaborative work between partners : P.J. TOPHAM, I.H. GOODRIDGE, D.V.A. BENN and A.J. HOLDEN (Plessey), T. KERR and P. REES (GEC), F. ALEXANDRE, E. CAQUOT and C. CHEVALLIER (CNET), B.N. LYONS and J. O'BRIEN (F.T.L.), P.N. KEMBER (P.T.).

8. REFERENCES

[1] P.J. Topham et al. "Heterojunction bipolar digital IC's using MOCVD material", GaAs IC's Symposium, 1986.

[2] Daoud-Ketata, Bresse, Dubon-Chevallier, "A self-aligned technology using refractory metals for GaAs-GaAlAs heterojonction bipolar transistor", GaAs and Rel. Comp. Conf., Las Vegas, 86.

[3] P. Desrousseaux, J. Dangla, M. Laporte, E. Caquot, D. Etiemble and A. Kazeminejad, "GaAlAs/GaAs Heterojunction bipolar transistor : Models and HECL circuits design", IEEE Bipolar Conference, 1987.

[4] C.G. Eddison, E.J. Greenwood, R.C. Hayes, P.T. Topham, D.V.A. Benn, GaAs Heterojunction Bipolar Transistor ECL Divide by 4 Circuit Operating at frequency greater than 5.6 GHz", ESSIRC, 1987.

[5] P. Ashbum A. Brunn-Schweller, A. Rezazadeh and E. Chor, "Comparison of silicon and GaAs/GaAlAs Heterojunction Bipolar Technologies for High-Speed ECL Circuits", IEEE Bipolar Conference, 1987.

[6] M. Madihian et al. : "Fabrication and modelling of a novel self-aligned AlGaAs/GaAs heterojunction bipolar transistor with a cutoff frequency of 45 GHz", IEDM 86 Proceedings, 1986.

[7] Ishibashi et al., "Self-aligned AlGaAs-GaAs heterojonction bipolar transistor for high-speed digital circuits", IEDM 86, Proceedings 1986, pp. 809-810.

Project No. 1128

IMPROVEMENTS IN GaAs MATERIAL FOR IC'S APPLICATIONS

Martin G.M., Deconinck P., Duseaux M. and Maluenda J.,
Nagel G.+ and Löhnert K.+,
Crochet M.J.*, Dupret F.* and Nicodème P.*

Laboratoire d'Electronique et de Physique Appliquée
3 Avenue Descartes, PB 15, 94451 Limeil-Brévannes, France

+ Wacker-Chemitronic, P.O. Box 8263 Burghausen, F.R.G.

* Unité de Mécanique Appliquée, Université Catholique de Louvain
2 Place du Levant, 1348 Louvain-la-Neuve, Belgium

1. INTRODUCTION

In recent years, ultra high speed LSI GaAs Integrated Circuits have been achieved in several laboratories all over the world. Getting high fabrication yield for such LSI devices requires extremely homogeneous wafers. The goal of the 1128 program is to obtain large crystals (diameter ≥3 inches) of GaAs material suitable for LSI application. More specifically, this program is aimed at developing an industrial approach for the preparation of large diameter (≥3 inches) semi-insulating GaAs substrates which can allow one to prepare very homogeneous active layers by ion implantation and thus to control the properties of each individual micro-FET made on it [1].

Our joint work, which has started in January 1985, may be divided into two main parts:

i. Liquid Encapsulated Czochralski (LEC) growth of GaAs ingots and detailed characterization which includes assessment of as-grown materials (Etch Pit Density, EL2 concentration, resistivity, mobility, ion implantation efficiency) and evaluation of homogeneity of active layers made on them by means of ion implantation (micro-FET Dense Row Pattern procedure described below);

ii. Computer simulation of LEC growth taking into account the different types of global heat transfer in actual pullers, and calculation of induced thermal stress field in the growing ingot.

In this paper, we will first review the progress which has been made over the last eighteen months on the growth of LEC crystals together with the specific techniques which have been developed for attaining homogeneous properties of the wafers. We will then elaborate on the specific measurement techniques which have been used for evaluating the quality of the material. Finally, we will present the basic principles of the numerical techniques which have been developed towards an accurate simulation of the growth progress and a deep understanding of the thermal exchanges taking place in the puller.

2. GROWTH AND CHARACTERIZATION OF SEMI-INSULATING GaAs SUBSTRATES

2.1 Growth of large diameter semi-insulating GaAs substrates

The manufacturing of LSI GaAs circuits requires high quality of the semi-insulating GaAs substrate material with respect to uniformity of the electrical properties. At present the quality of undoped dislocated and non-annealed GaAs substrate grown by conventional LEC technology has proven to be insufficient for meeting these requirements. Fluctuations of the threshold voltage in such material have been directly correlated to the structure of the built-in dislocation network and associated non-uniformities in the concentration of the deep level EL2. Recent results indicate that various techniques such as post-growth annealing [2-7], magnetic field crystal growth [8,9], accurate stoichiometry control [10,11] and In-alloying [12-14] lead to high uniformity of the electrical properties.

In order to achieve the best approach for growing high uniformity GaAs substrates, we have investigated two main routes:
i. growth of low dislocation density material by flattening the thermal gradients in accordance with the results of LEC modeling and/or by In-alloying;
ii. improvement of uniformity of the electrical properties through the application of a magnetic field and post-growth annealing techniques.

2.1.1 Experimental

Two inch diameter undoped and In-alloyed and three inch diameter undoped GaAs crystals investigated in this paper were grown by the low pressure LEC technology without and with the application of a vertical magnetic field (3" diameter only with a range of magnetic field from 0 to 2500 Gauss). The ingots were partially post-growth annealed in evacuated and sealed quartz ampoules at temperatures above 1000°C with additional arsenic. The ingots have been characterized by EPD etching, x-ray topography for evaluating the dislocation density, resistivity mapping by the point contact method (spot size 200 μm, spacing of spots 400 μm) and by IR absorption measurements (linescan profiles, spot size 300 μm) at a wavelength of 1 μm for evaluating the EL2 concentration.

2.1.2 Reduction of the dislocation density

The reduction of the dislocation density may be obtained either by flattening the thermal gradients (e.g. improvement of the growth parameters and/or the thermal environment) or by isoelectronic doping with Indium. Consequently the thermal stresses in the crystal are lowered or the critical shear stress is increased while the formation of dislocations is reduced. In figure 1 we show the radial EPD distribution of a 2" diameter Cr -doped semi-insulating GaAs wafer. The average dislocation density (EPD according to ASTM standard F47) in this wafer is 14500 cm^{-2}; this is significantly lower than in standard grown ingots. This improvement was obtained by a well defined cone angle of the crystal which reduces the heat exchange between cone surface and the thermal environment and results in low thermal gradients in the crystal. On the other hand, In-alloying is very efficient in eliminating dislocations as one can clearly see on the x-ray topography in figure 2. The wafer is completly dislocation-free except at the edge where one finds residual slip lines.

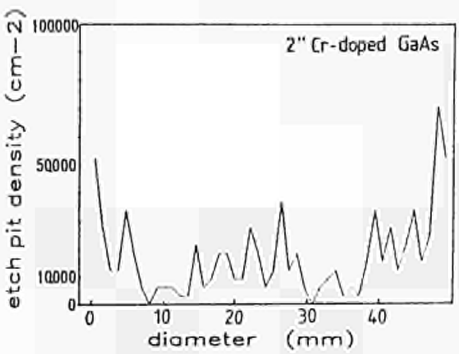


Figure 1. Radial Etch Pit distribution of a 2" diameter Cr-doped semi-insulating GaAs ingot grown with improved LEC.

Figure 2. X-ray topograph of a 2" diameter In-alloyed semi-insulating GaAs wafer.

2.1.3 Post growth annealing

Post growth ingot annealing has proven to be a very effective tool for obtaining excellent uniformity of the electrical properties. Figure 3 shows typical linescan profiles of the IR absorption coefficient across wafers from the same ingot before and after ingot annealing. The as-grown state (a) clearly exhibits the familiar W-shaped profile with strong microscopic fluctuations, which is directly related to the distribution of dislocations and their microscopic arrangement in a cellular network. After ingot annealing (b) the W-shaped profile is largely flattened and fluctuations are reduced to below ±9% of the mean value, which corresponds to a relative standard variation of less than 3%. In addition it can be seen from figure 3a and 3b that the average value of the absorption coefficient and thus the EL2 concentration increases after ingot annealing.

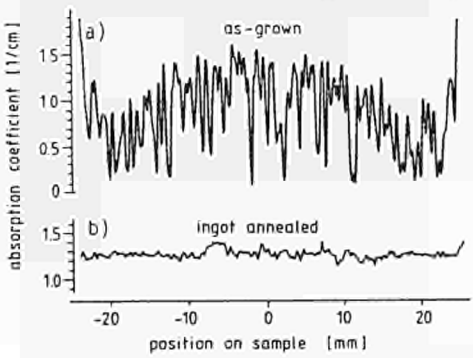


Figure 3. IR absorption profiles of 2" diameter undoped GaAs wafers in the as-grown state (a) and after ingot annealing (b).

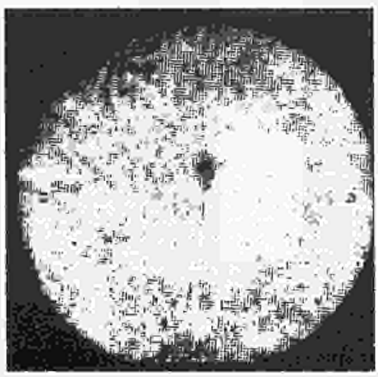


Figure 4. Resistivity mapping by the point contact method on a 3" diameter undoped GaAs wafer (ingot annealed).
The difference between the dark and white spots corresponds to a deviation of ±30 % of the mean value of resistivity .

In figure 4 we show the resistivity mapping of a 3" diameter undoped GaAs wafer after ingot annealing. The difference between the white and dark spots is in the range of about ±30% of the mean value (standard deviation: 10%).

For comparison we show in figure 5 the radial profiles of the EL2 concentration of 2" diameter In-alloyed semi-insulating GaAs wafers. As expected already from figure 2 (x-ray topograph) the elimination of the dislocation cell structure achieved in this material results in very good uniformity of the EL2 distribution already in the as-grown state. Consequently additional ingot annealing does not provide further improvement of the uniformity but as in the case of dislocated material the average EL2 concentration is also increased.

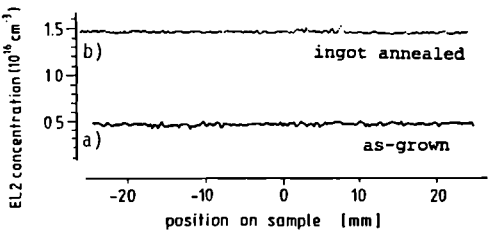


Figure 5. Radial EL2 concentration profiles (seed and wafers) in as-grown (a) and ingot annealed (b) 2" In-alloyed nearly dislocation-free GaAs crystals (resolution 300μm).

2.1.4 Investigation of uniformity by applying Vertical Magnetic field LEC technique

EL2 concentration profiles of wafers from the middle part of a 3" diameter GaAs crystal in the as-grown state are shown in figure 6. During growth of this part the magnetic field was switched on and then maintained at 2500 Gauss. One can clearly see that the EL2 concentration profiles of the wafers from the LEC and adjacent VM-LEC grown part are nearly comparable with respect to long range and short range variations (sEL2 = sigma EL2/<EL2> = 5%, <EL2> = average concentration). This means that for these growth conditions the magnetic field does not provide an improvement over the standard LEC growth. It should be pointed out, however, that for the as-grown state the EL2 uniformity of this ingot is exceptionally good and already comparable to typical ingot annealed 3" diameter material.

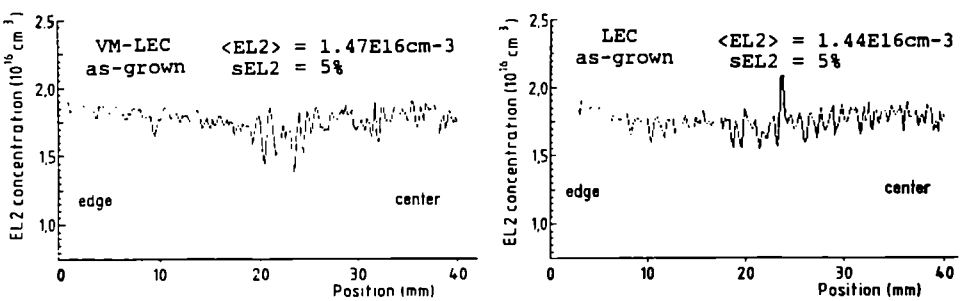


Figure 6. Radial EL2 concentration profiles of wafers from a non-annealed 3" diameter undoped ingot temporarily grown with (a) and without (b) a vertical magnetic field (resolution 300 μm).

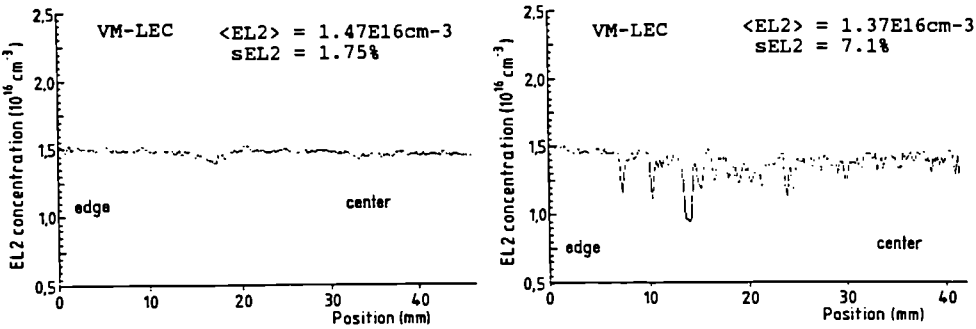


Figure 7. Radial EL2 concentration profiles of seed end (a) and tail end (b) wafers from one VM-LEC grown (600 Gauss) 3" diameter ingot annealed crystal (resolution 300 μm).

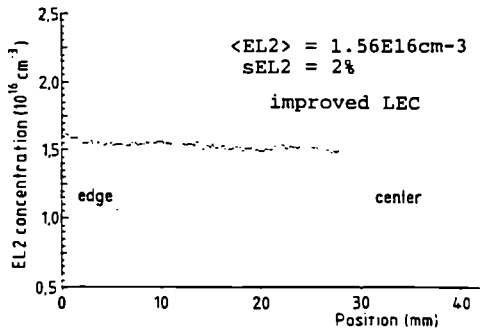


Figure 8. Radial EL2 concentration profile of an ingot annealed 3" diameter undoped crystal (seed end wafer) grown with improved LEC technology (resolution 300 μm).

The results of the EL2 concentration profiles in seed and tail end of an ingot annealed VM-LEC grown crystal are shown in figure 7a and 7b. The seed end wafer (see figure 7a) reveals an excellent uniformity (sEL2=1.75%) which is clearly better than typical results in standard LEC grown ingots. Similar results can be obtained by optimized LEC growth without magnetic field (see figure 8, sEL2=2%). The EL2 concentration profile of the tail end wafer (figure 7b) exhibits a moderate uniformity with remaining fluctuations. In order to obtain the same excellent uniformity both in seed and tail end, a precise control of the initial melt stoichiometry near the congruent melting point is required; this can be achieved by Arsenic Injection into the melt. This will be investigated in the future.

2.2 Characterization by micro fet analysis (dense row pattern process)

The spread of electrical properties of IC's active layer on a wafer may strongly decrease the yield of fabrication of LSI devices. More specificaly we must consider the spread of properties of two neighbouring FET's which can be only a few microns spaced from each other in this type of circuit.

This is the case in static memories where the unit cell presents two drivers FET's only 10μ m apart from each other (figure 9). A difference in their threshold voltage leads to an instability of the cell and hence the misworking of the entire memory. The uniformity of wafers is thus of prime importance.

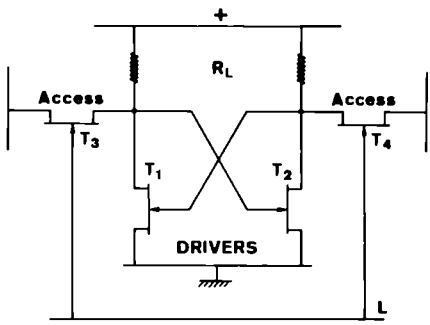


Figure 9. Schematic of a memory cell.

To test the material we have used the so-called DRP (Dense Row Pattern) procedure established together by LEP and SIEMENS within the frame of ESPRIT 843. It consists in measuring the fluctuation of the threshold voltage of thirty micro FET's, 5 micrometers far from each other displayed all over the wafer (figure 10). Of course such a highly dense packing of FET's simulates actual situations in LSI circuits. This pattern is repeated more than 260 times all over a 2" wafer and more than 640 times over a 3" wafer in two perpendicular directions. The process has been reduced to its minimum (set of five levels of masks) so as to get rid of any technological effect.

Many 2" and 3" wafers from different parts (seed and tail) of different ingots grown by WACKER have been tested (Cr-doped, undoped, In-alloyed, low thermal gradient growth or standard thermal gradient growth). For each type of wafer one measures all the FET's and their threshold voltage Vth. From these data, we calculate:

- the standard deviation of the thirty FET's in the same row (σVth) and the corresponding mean value of all the rows on the wafer;
- the spread around the mean value ($\sigma(\sigma Vth)$);
- and, more important, the percentage P15 of dense rows with a standard deviation below 15mV.

Figure 11 shows the distribution curve of σVth for an In-alloyed, a Cr-doped material and an undoped material. Table 1 reports the results for the different materials. According to a LEP evaluation done outside this contract the value of P15 must be at least equal to 75% so as to get a material related yield of LSI circuits (a 1K-SRAM taken as a pratical example) close to 100%. Thus today at least one type of 2" material meets this requirement.

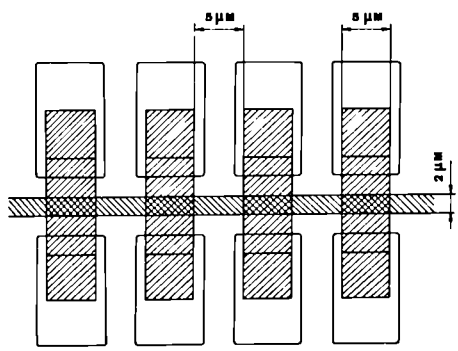


Figure 10. Schematic of a part of the dense row of 30 micro-FET's.

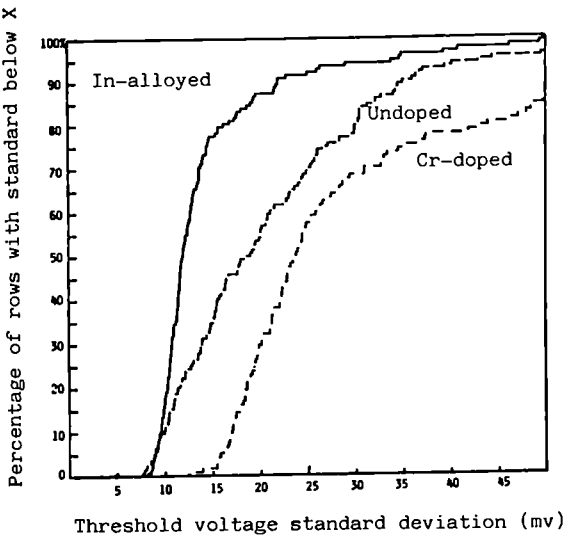


Figure 11. Distribution curve of σVth for In-alloyed undoped and Cr-doped materials.

Table I. Characterization results obtained with various materials.

DIAMETER	2"						3"	
MATERIAL	CR-DOPED		UNDOPED		IN-ALLOYED		UNDOPED	
INGOT n°	C/61831		U/01931		I/89231		U/68831	
Seed or tail	S	T	S	T	S	T	S	T
EPD (cm^{-2})	35000	29000	18000	20000	<500	<500	29000	37000
ELECTRICAL CHARACTERISATION								
Mobility (cm2/V.s)	3533	2283	6977	6431	2700	3010	6441	5824
Resistivity(Ohm.cm)	3.5E8	5.6E8	4.1E7	4.2E7	8.0E7	1.2E8	5.1E7	7.7E7
HOMOGENEITY FROM DRP :								
σ(Vth) (mV)	31	44	23	27	15		30	
σ(σVth) (mV)	18	30	20	43	8		46	
% of σ <15 (mV)	6	0	35	38	75		28	

2.3 Investigations by high resolution tomography

Recently a tomography system whose principle is described in figures 12a and 12b has been designed for high resolution imaging: light produced by a YAG laser and entering the cleaved edge of a wafer is diffused by the defects present in the material. Agregates or precipitates have been evidenced by this new technique (figure 13) gathering along lines (dislocation lines probably) or in clouds. It becomes possible, with this new technique, to locate defects in the three dimensional space by monitoring the position of the laser beam. This new technique is expected to become important to qualify ingots and/or wafers, since it is the first to provide an easy evaluation of the presence of precipitates.

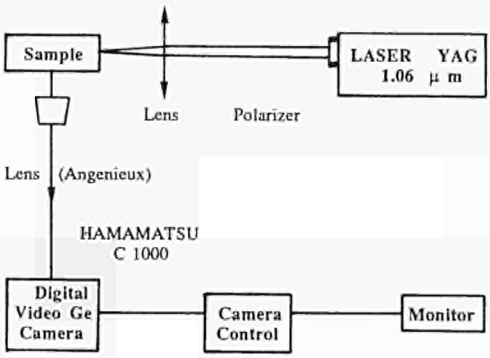


Figure 12a. Tomography apparatus.

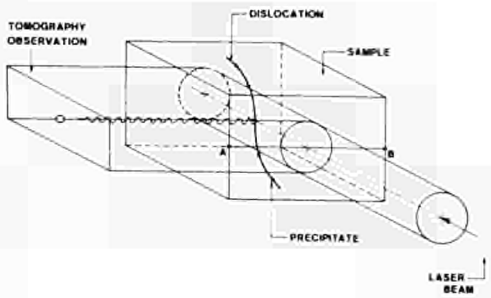


Figure 12b. Principle of high resolution tomography.

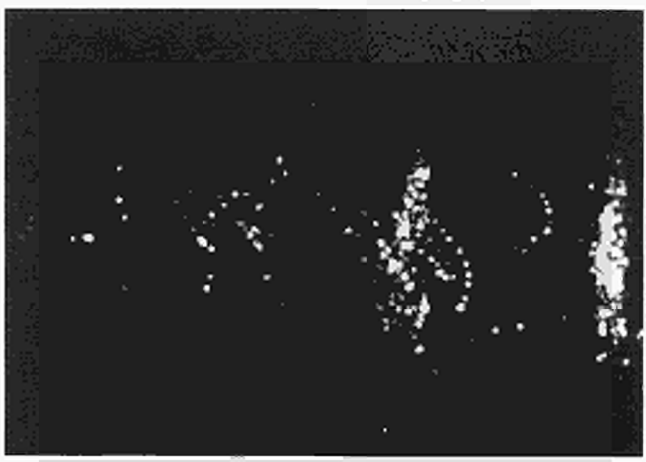


Figure 13. Infrared tomography picture taken with a high magnification and a YAG-laser beam about 100 μm in diameter. The total length of the scanned material is here 1 μm.

3. LEC MODELING

3.1 Global finite element analysis

Any improvement of the processing conditions for growing GaAs material suitable for LSI application requires an accurate knowledge of the thermal stress distribution in the growing crystal and of the liquid/solid interface. Such knowledge requires the development of a detailed model describing the heat transfer within the furnace. The problem is highly complex since it involves radiation, heat conduction, heating elements and convection-diffusion within the melt. As a basis for our work in the present project, we have used a global finite element analysis of the Czochralski furnace [15] which we now briefly review before commenting on the new specific features which have been added for calculating the growth of GaAs crystals.

Let us consider in Fig 14a the draught of a Czochralski puller in which one can easily identify the resistor R, the insulator I, the crucible C, the melt M, the crystal G, the structure S and the dome D. The numerical method consists of a separate analysis of each individual component of the crucible by means of the finite element method [16] followed by an assembly of these components which takes radiative and convective exchanges into account. The aim of the algorithm is to obtain a sequence of linearized systems in terms of the temperatures at the nodes of the finite element mesh. Each *solid body* within the furnace, such as R, I, C, G and S in Fig 14a, together with the melt M as long as forced and natural convection is not being calculated, is covered by a finite element mesh in which one takes conduction and possibly heat generation into account. The right-hand side of Fig 14a shows a finite element mesh used in a typical calculation. We note that the shape of the crystal G and of the melt M are unknown *a priori* because the shape of the liquid/solid interface is unknown at the outset.

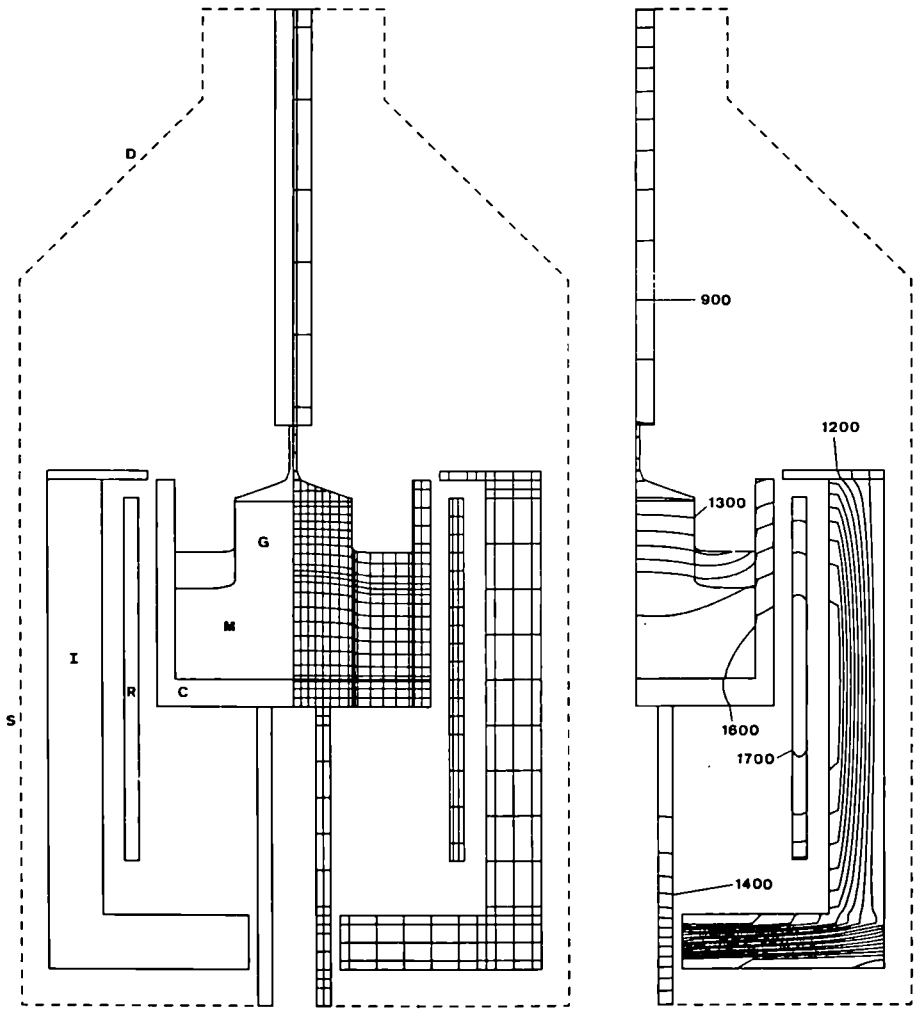


Figure 14. (a) Typical Czochralski puller with the resistor R, the insulator I, the crucible C, the melt M, the crystal G,the structure S and the dome D and the finite element mesh; ; (b) isotherms in the puller.

Throughout the iterative process, the finite element mesh is deformed in order to adapt to the latest interface shape. Some parts of the furnace such as the dome D in Fig 14a are *thin bodies* which are best represented by one-dimensional elements on which one imposes a relationship between the outgoing heat flux and the local temperature.
Finally, the furnace contains several *radiative enclosures* which are three-dimensional axisymmetric domains connecting the solid and the thin bodies described above. The most complex part of the thermal problem is the calculation of the radiative exchanges taking place in the enclosures. The calculation is of course highly non-linear in view of the fourth power terms in temperature appearing in Stefan's law; moreover, for calculating the radiative flux at a given point of an enclosure, one must take into account the hidden and viewed parts of the furnace from that point [17] [18].

Each of these constituent parts, i.e. the solid bodies, the thin bodies and the radiative enclosures, are called *macro-elements* . The boundaries of these macro-elements form the *skeleton* of the furnace. After an assembly of the macro-elements, one imposes the thermal equilibrium and the uniqueness of the temperature on the skeleton of the furnace. The resulting non-linear system is solved by means of Newton's method. The finite element mesh is then updated in such a way that the liquid/solid interface coincides with the melting point isotherm.

3.2 Specific features of GaAs crystal growth

The method described in the previous section requires a precise knowledge of the geometry of the Czochralski puller and of the physical properties of its constituents. The geometrical data also include the radius of the crystal, the level of the melt in the crucible, the relative altitude of the latter with respect to the heater, etc... Once the parameters of the furnace have been introduced as an input for the calculation, one is able to obtain a relationship between the heating power and the pulling rate, with the additional constraint that the triple point at the intersection between the liquid, solid and gaseous phases be assigned the melting temperature. In practice, one imposes a pulling rate based on experimental data; the resulting calculated input power and the shape of the isotherms throughout the furnace allow one to verify the validity of the mathematical model.

The features described in the previous section are insufficient for an accurate description of LEC growth. Indeed, the melt is covered by a layer of boric oxide to avoid the arsenic evaporation. Moreover, the high pressure argon atmosphere above the boric oxide layer generates further convective heat exchange in the furnace. The global finite element model has thus been refined for including such features. The constituents of the furnace indicated in Fig 14a were all either fully transparent either fully opaque. The boric oxide layer, indicated by the symbol B in Fig 14a, is neither transparent nor opaque. It is a semi-transparent material; absorption does not solely take place at the surface of the body but energy is also absorbed and emitted within the body itself. For taking such properties into account, we have assumed that the boric oxide is opaque for some ranges of wavelength and transparent for some others. One may then obtain, on the basis of experimental data, an *opacity coefficient* which is assumed to be temperature independent. Despite the presence of the boric oxide layer, the surface of the melt is then exchanging energy by radiation with the upper constituents of the furnace.

The convective motion of the ambiant gas in the enclosures of the furnace depends upon the value of the Grashof number, which compares tbuoyancy to tviscous forcesc. For actual values of the Grashof number in the GaAs furnaces, one finds thin boundary layers along the walls of the cavity while gas temperature outside these layers is essentially uniform. We have developed a mathematical model based upon that observation.

Let q denote the heat flux at the surface of an enclosure, T its temperature, and let T^* be the (assumed) uniform temperature away from the boundary layers. We impose a relationship

$$q = m\,(T - T^*),$$

where m is a coefficient depending upon the Grashof and the Prandtl numbers which can be calibrated from specific calculations on the motion of the ambiant gas in the enclosure . For further details on the evaluation of m, the reader is referred to [19] [20].

As a result of the calculation, one can then obtain a full description of the temperature field throughout the furnace constituents.

An example of such a temperature field is shown in Fig 14b, which shows the isotherms in the solid parts of the furnace under standard conditions when the cristal has a diameter of 2.5 inches.

3.3 Calculation of thermal stresses

The presence of dislocations in the grown crystal is attributed in part to excessive thermal stresses being generated during growth. The usefulness of the calculations described above is therefore enhanced by a method for monitoring the thermal stresses while the crystal is being pulled. A major problem which needs further research is of course the selection of an appropriate continuum model for calculating the thermal stresses and the generation of dislocations, which is made difficult by the lack of data about the mechanical behavior of GaAs near the melting point.

It is currently assumed that the crystal behaves as an axisymmetric thermoelastic solid. The precise knowledge of the temperature field based on the global calculation allows one to use once more the finite element technique for calculating the thermal stresses. The calculation, which is fairly standard, assumes that surface forces vanish on the external surface of the crystal. It is then possible to calculate the principal stresses S_1, S_2 and S_3 at every point of the crystal and to evaluate Mises invariant which has the following form,

$$S_M = [\,(S_1 - S_2)^2 + (S_2 - S_3)^2 + (S_3 - S_1)^2\,]^{1/2};$$

S_M is currently adopted as an indicator for the generation of dislocations. An optimization of the furnace should lead to lower values of the Mises invariant, for a fixed value of the pulling rate and of the diameter of the crystal.

3.4 Application

As a typical application of the method described in earlier sections, we present below results which have been obtained in simulating the pulling of 2.5" GaAs crystals at three different pulling rates, i.e. 0, 1.0 and 2.0 cm/h. All the specific features of the method have been taken into account; we have also included the modified shapes of the menisci due to surface tension. Fig 15 shows the temperature field within the melt, the crystal and the boric oxide layer together with the thermal stresses in the crystal (in MPa). One observes that a higher pulling rate induces a higher concavity of the crystal near the liquid/solid interface, a weaker temperature gradient in the melt below the interface but stronger above and an increase of the thermal stresses within the crystal, as one may expect. One also calculates that a higher pulling rates requires a lower power dissipation in the resistor.

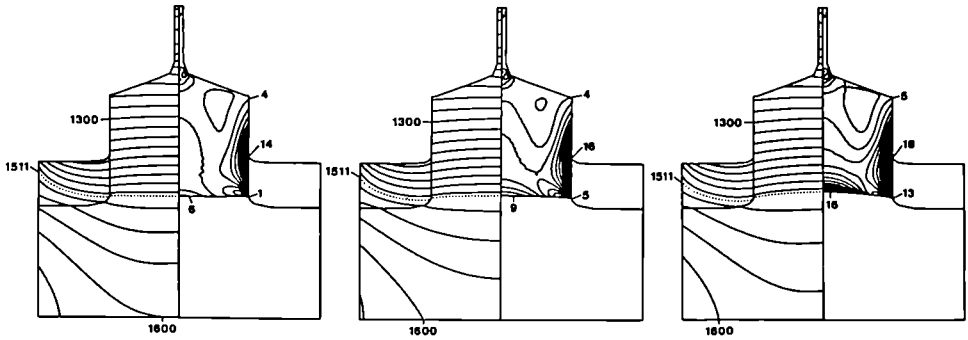


Figure 15. Temperature field within the melt, the crystal and the boric oxide layer together with the thermal stresses in the crystal (in MPA) for three different pulling rates: 0, 1.0 and 2.0 cm/h.

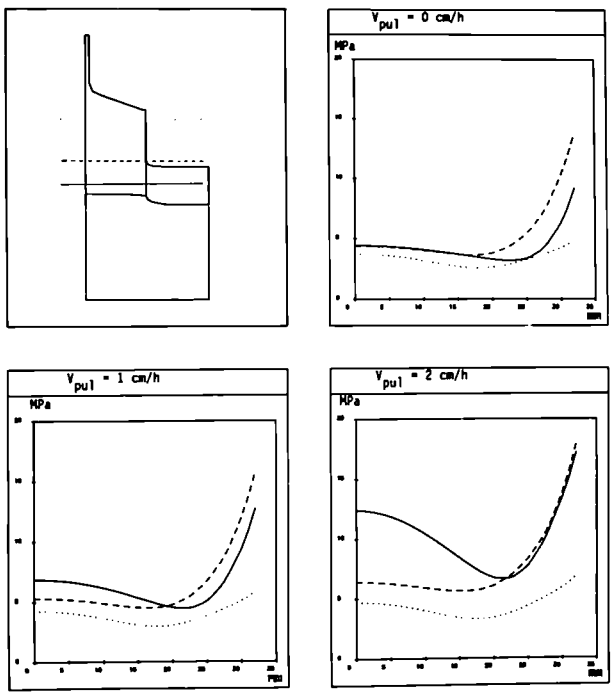


Figure 16. Radial distribution of thermal stresses in the crystal at three different altitudes for the pulling rates of figure 15.

In Fig 16, we show the radial distribution of thermal stresses in the crystal at three different altitudes and for the cases of Fig 15. We observe that the stresses are higher near the wall of the crystal, where the thermal gradients are not uniform. When the pulling rate increases, one observes that the stresses increase on the axis of symmetry. In fact, when the pulling rate is 2cm/h, the minimum value of S_M is not found on the axis of symmetry.

4. CONCLUSIONS

Manufacturing of high quality GaAs substrate material has been realized by different routes such as lowering of the dislocation density (improved thermal gradients or In-alloying) and improving of the uniformity by the application of magnetic field technique and post growth annealing. The present results indicate that within the probing resolution used, excellent uniformity of the EL2 concentration, comparable to In-alloyed GaAs can be achieved in undoped material by magnetic field or improved LEC technique and additional ingot annealing.

For improving the thermal conditions during crystal growth, a powerful numerical method for calculating the global heat transfer in a Czochralski furnace was developed. At present it is already possible to determine optimal operating conditions and optimal geometry with a view to lower thermal stresses in the crystals. In future the effect of forced and natural convection within the melt will also be included.

Wafers have been evaluated by the Dense Row Pattern procedure with respect to their suitability for manufacturing of LSI circuits (e.g. 1k-SRAM). This technique allows one to determine the micro uniformity of FET properties and to evaluate the material-related yield of functional IC's on the wafer. These requirements of LSI circuits are met today by 2" In-alloyed material. Furthermore high resolution tomography has been described as a promising technique, which may become important to qualify ingots and wafers with respect to defects such as agregates and/or precipitates.

REFERENCES

[1] Maluenda J., Martin G.M., Schink H. and Packeiser G. (1986). Appl. Phys. Lett.48, 715
[2] Rumsby R., Ware R.M., Smith B., Tyjberg M., Brozel M.R. and Foulkes E.J. (1983) Tech. Digest. GaAs IC Symp., Phoenix Az., 34
[3] Miyazawa S., Honda T., Ishii Y. and Ishida S. (1984) Appl. Phys. Lett. 44, 410
[4] Martin S., Duseaux M. and Erman M. (1984). Inst. Phys. Conf. Ser. 74, 53
[5] Yokogawa M., Nishine S., Matsumoyo., Morishita H., Fujita K. and Akai S. (1984). Inst. Phys. Conf. Ser. 74, 29
[6] Chin A.K., Camlibel I., Caruso R., Young M.S.S. and Van Neida A.R. (1985). Appl. Phys., 57, 2203
[7] Löhnert K., Wettling K. and Koschek G. (1986). *Semi-insulating III-V-Materials* (1986) (Ohmsha Ltd., Tokyo), 267
[8] Terashima K., Yahata A. and Fukuda T. (1986). J. Appl. Phys. 59, 982
[9] Kimura T., Katsumata T., Nakajima M. and Fukuda T. (1986). J.C rystal Growth 79, 264
[10] Katsumata T., Okada H., Obokata T. and Fukuda T. (1987). J. Appl. Phys. 61, 1469
[11] Inada T., Sato T., Ishida K., and Fukuda T. (1986). J. Electron. Mater. 15, 169
[12] Ohmori M. (1984). Inst. Phys. Conf. Ser. 74, 647
[13] Hyuga F., Kohda H., Nakanishi H., Kobayashi T. and Hoshikawa K. (1985). Appl. Phys. Lett. 47, 620
[14] Löhnert K., Nagel G. and Wettling W. (1986), *Proc. Advanced materials for telecomunication*, Strasbourg (Les éditions de Physique, Les Ulis), 65
[15] Dupret F., Ryckmans Y., Wouters P. and Crochet M.J., (1986), J. Cryst. Growth 79, 84
[16] Zienkiewicz O.C., *The Finite Element Method*, (1977), Mc Graw-Hill, New-York
[17] Wouters P., *Simulation Numerique des échanges thermiques et application à la croissance des cristaux semi-conducteurs*, (1985), Thèse de doctorat, Université Catholique de Louvain, Louvain-la-Neuve.
[18] Siegel R., Howell J.R., *Thermal radiation heat transfer*, (1981) Mc Graw-Hill, New-York
[19] Jordan A.S., (1980), J. Cryst. Growth 49, 631
[20] Mc Adams W.H., *Heat Transmission*, (1954), Mc Graw-Hill, New-york

Project No. 833

LARGE AREA COMPLEX LIQUID CRYSTAL DISPLAYS ADDRESSED BY THIN-FILM TRANSISTORS

M.G. CLARK, P. MIGLIORATO, N.J. BRYER, P.A. COXON

GEC Research Limited, East Lane, Wembley, Middx. HA9 7PP

J. MAGARINO, J.P. LE PESANT

Thomson-CSF, Laboratoire Central de Recherches,
Domaine de Corbeville, BP10, Orsay, France.

W. SENSKE, K.H. GREEB, K.FAHRENSCHON

AEG Forschungsinstitut, F1 33-F, Goldsteinstrasse 235,
Frankfurt, W. Germany.

F. MORIN,

ROC, CNET, Route de Tregastel, BP40, 22302 Lannion Cedex, France

M.B. ANDERSEN,

A/S Modulex, Kooeverveg 101, DK-7190 Billund, Denmark.

The next generation of office systems and portable computers will require a flat electronic display of A4 size. Thin-film transistor addressing of liquid crystal displays would allow an increase in display size and complexity to those required for word processor and graphic displays. The aim of this project is to assess the feasibility of making such a display and producing it competitively. We report here the excellent results which have been obtained with both amorphous and polycrystalline silicon thin-film transistors and present the incorporation of both colour and user-interaction into active-matrix addressed displays. Exploitation of these achievements will put Europe in a strong position to compete with established Japanese and American firms in this rapidly expanding market.

1. INTRODUCTION

In this paper we report on the results of an 18 month feasibility study for the production of large area high information content liquid crystal displays using silicon TFT active matrix addressing. The ultimate target is the definition of a production process for A4 sized display with an information capacity equivalent to that of the printed page (about 5500 alpha-numeric

characters) after a further 3 years. Equipment for handling A4 sized displays is not readily available for all processing steps at present, but, driven by the large world wide interest in this type of display, equipment manufacturers have suitable equipment at an advanced stage of development, and two manufacturers have been sub-contracted to produce prototype machines as part of this project. For this reason work at this stage has concentrated on making smaller size demonstration displays to develop the fabricaton process whilst studying the problems of processing A4 sized substrates in parallel.

A more general background to the project was presented in last year's paper [1] in which we reported important progress in TFT performance and the fabrication of small test displays. We are investigating the use of both amorphous and polycrystalline silicon TFTs for the active matrix and good results have been achieved for both. An inverted staggered structure for amorphous silicon TFTs was identified as the preferred one, and ON/OFF current ratios of ~ 10^8, carrier mobilities of 0.7cm^2/s and threshold voltages of ~2V were achieved. We were also able to report a new low pressure CVD (LPCVD) polysilicon deposition process which produced films of superior material properties to conventional LPCVD films and this breakthrough enabled the fabrication of polysilicon TFTs with an ON/OFF current ratio of 5 orders of magnitude, carrier mobility of 8-10cm^2/Vs and threshold voltage of 8V. The relative merits of the two types of transistor were discussed. A new display circuit configuration which simplifies the fabrication process and eliminates drive line cross-overs on the active matrix back-plane (and hence the possibility of associated short-circuit defects) was also described.

This year we are able to report a further modification to the display circuit configuration which eliminates the possibility of defective lines in the display due to transistor faults. Results on TFTs fabricated in polysilicon deposited at ultra-low pressures showing greatly improved performance will also be presented. Displays with 224 x 208 pixels and a viewing area of 89x81mm^2 using polysilicon TFTs and the new fault- tolerant circuit configuration have been fabricated.

Work has progressed on the a-Si technology and displays with 256x320 pixels over 65x80mm^2 have been produced. Improvements in all assembly techniques and interconnections for TFT-matrix LCDs are also reported.

The results of work on two important added features for the display are presented. Firstly the implementation of colour; a 320x320 pixel demonstration display using a technology compatible with the TFT matrix-addressed liquid crystal display is presented. Secondly, an interactive display over 10x13cm^2 with a resolution of 0.4mm has been produced. This is an important step forwards towards the development of the so-called electronic paper which is one of the specific objectives of the ESPRIT programme.

2. ARRAY TECHNOLOGY

2.1 Amorphous Silicon

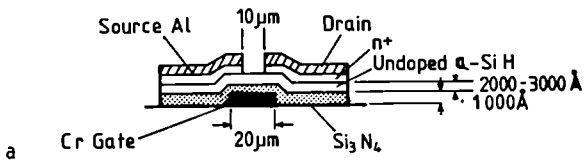
Amorphous silicon deposition and thin film transistor fabrication is now a well-established technique which was described at length in the 1986 ESPRIT Technical Week paper. We recall here the process steps which have been

developed during this phase of the project: Figure 1a shows a section through an individual inverted staggered TFT and Fig.1b illustrates the design of a pixel in a liquid crystal.

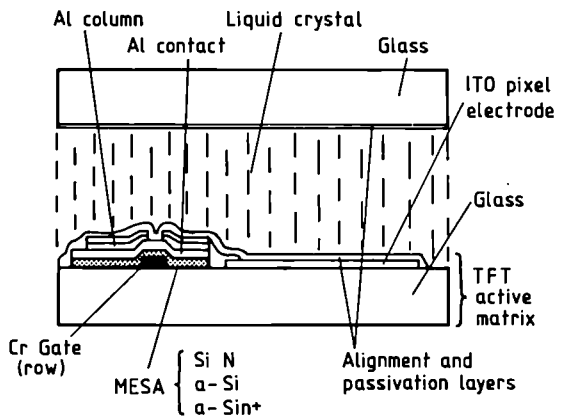
The fabrication steps we used are the following:

1) Deposition and etching of the ITO pixel electrode on an ordinary glass substrate (soda-lime glass).
2) Deposition and realization of the line and gate electrodes using Cr,Al or both.
3) Deposition of Si_3N_4 undoped a-Si:H and n+- doped a-Si:H in the same run and at the same temperature (T_p=250°C) by plasma-enhanced CVD using pure SiH_4 and NH_3 for the insulator.
4) Amorphous silicon or both a-Si:H and Si_3N_4 are etched to form the transistor.
5) Contact holes are etched through the nitride down to the ITO for connection to the drain contact if necessary.
6) Top metallization to form the source, drain and column electrodes using Al,Cr or both and etching of n^+ doped a-Si:H.
7) A passivation layer using Si_3N_4 is deposited before adding the polyimide alignment layers.

A similar TFT technology has been established by AEG using SiO_2 deposited by PECVD for both gate insulator and final passivation layer. No significant differences in TFT characteristics have been found.



a



b

Figure 1: a)Cross-section of an inverted staggered a-Si TFT.
b)Cross-section of an a-Si TFT in an LCD pixel.

Four or five photolithographic steps are necessary to realize the TFT matrix. The devices have a 10µm channel length and a 20µm channel width for a pixel pitch of 250µm. Uniform deposition can be realized on 6 substrates of 15x15cm^2 in the same run on the plasma enhanced CVD reactor.

2.2 Polycrystalline silicon

In last year's paper GEC reported a new deposition process for polysilicon films using lower pressures (40mTorr as compared to 200mTorr typically used) This new process provided a breakthrough in controlling the film morphology resulting in larger grains of far greater crystal perfection. This is important as grain boundaries and crystal defects can give rise to trap levels in the semiconductor energy gap which in turn have a detrimental effect on the electrical effect on the electrical characteristics of polysilicon TFTs fabricated in the films. Figure 2 shows transfer characteristics for polysilicon TFTs fabricated in conventional polysilicon, hereafter referred to as type 1, and low pressure (40mTorr) deposited films, hereafter referred to as type 2. We had also achieved further improvement in device performance by using plasma hydrogenation as also shown in figure 2.

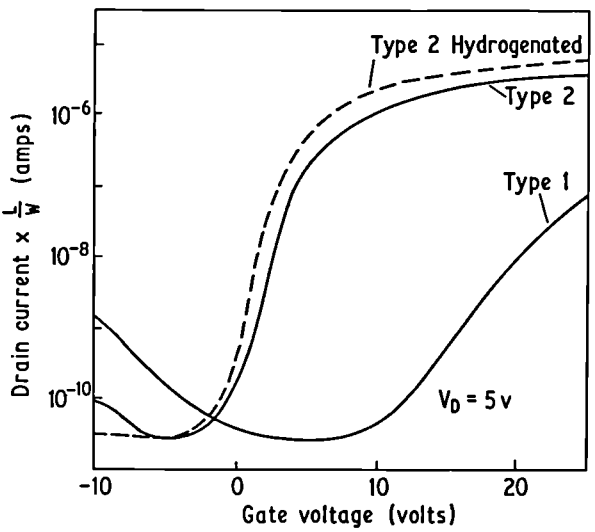


Figure 2: Polysilicon TFT transfer characteristics, t_{ox}=1000A.

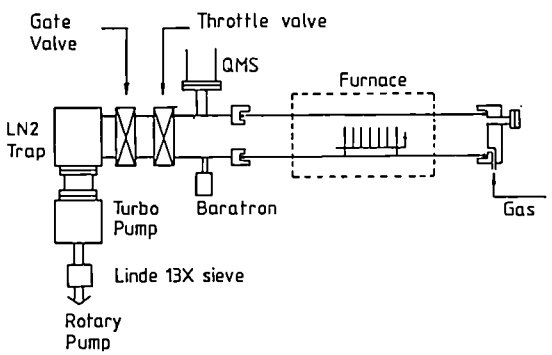
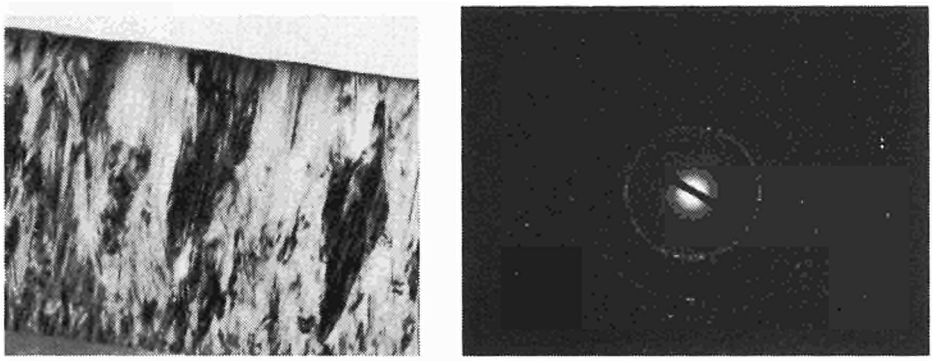
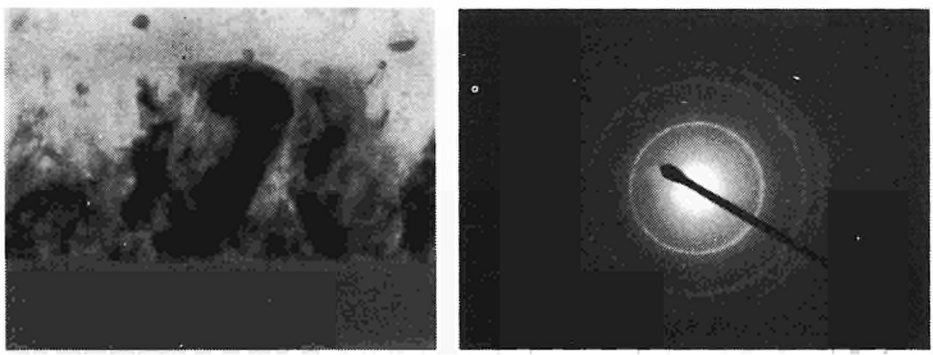


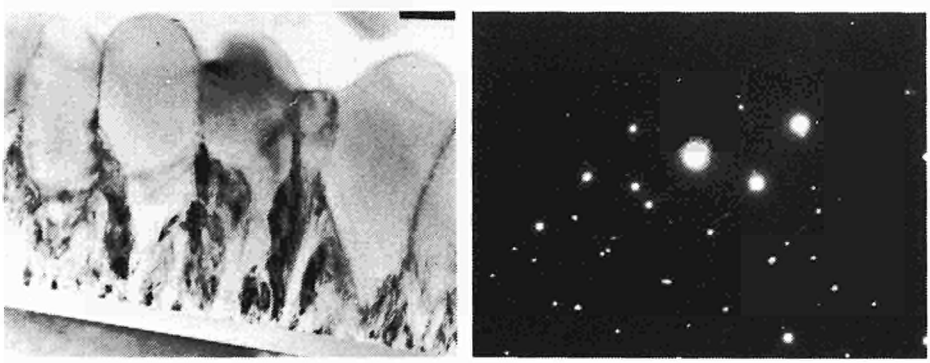
Figure 3: Schematic of in-house built LPCVD system.



TYPE I



TYPE II



TYPE III

Fig. 4. Transmission electron micrographs and electron diffraction patterns for the three types of material.

We also reported that a new in-house built LPCVD system, shown schematically in figure 3, which permits operation at pressures down to 0.1mTorr has been commissioned, and that films grown at ~ 2mTorr have much larger grains (0.3-0.5μm) showing no microtwinning or lattice distortion. In this paper we present initial results of TFTs fabricated in this type of material, henceforth referred to as type III.

Figure 4 shows transmission electron micrographs of the three types of material together with their associated electron diffraction patterns. The large grains and high crystal perfection in type III films is evident.

Figure 5 shows the transfer characteristic for a TFT fabricated in type III material. Typically an OFF-current less than $2x10^{-11}$A is achieved for a unity aspect ratio device with V_{DS} = 5V. The threshold voltage is ~8 V and the carrier mobility ~30 cm^2/Vs, The fabrication process was as outlined in last years paper.

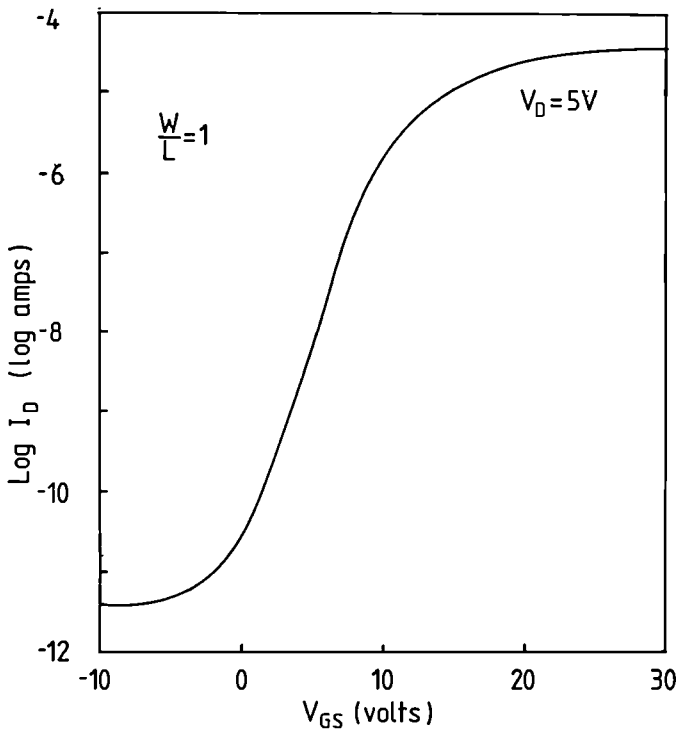


Fig.5 Transfer characteristic for a TFT fabricated in type III material.

The devices have a 0.1μm thick SiO_2 gate dielectric deposited by atmospheric pressure CVD, and have been plasma hydrogenated. We have started the work necessary to design a large area (A5-A4) polysilicon reactor. To this end we have employed our in-house built reactor to study the kinetics of polysilicon at these very low pressures, with parallel work on the morphology of the films to achieve the best compromise between film quality, likelihood of contamination, growth rate, and ease of achieving uniformity.
These results will be reported elsewhere (2).

3. NEW CIRCUIT CONFIGURATIONS

In last year's paper GEC reported a new circuit congfiguration, shown in figure 6 which elimiates drive line cross-overs in the active matrix backplane This eliminates the possibility of short circuit at cross-overs on the back-plane and simplifies the fabrication process as only one level of metallisation is required -hence we refer to this as the SLM (single-level-metallisation) circuit. In this paper we report on a further circuit reconfiguration we refer to as the capacitively-coupled-transistor (CCT) active matrix. In this con-figuration shown in figure 7, the pixels are now divided into two liquid crystal capacitors connected in series by the TFT. The pixel TFT is then capacitively coupled to a data and a reference line on the cell top plate, and only gate lines remain on the back plane. This simplifies the fabrication of the active substrate and eases the problem of edge connections. Such a simplification is achieved at the expense of the second plate which is now marginally more complex since it contains twice the number of tracks. However since the construction of the second plate is by far simpler, such a redistri-bution of complexity can only be beneficial in view of the final yield and costs.

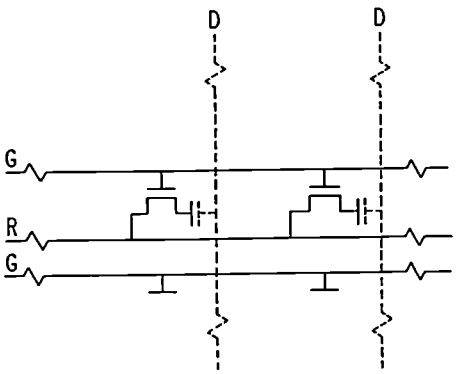


Fig. 6. SLM circuit configuration.

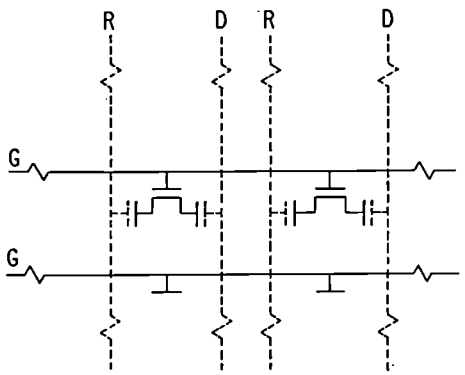


Fig. 7 CCT circuit configuration

The prinicipal advantage of this circuit is, however, that because the TFT is capacitively, rather than directly, coupled to the data and reference lines a short circuit with the gate line due to a defective TFT only affects the associated pixel rather than the entire row as is the case with the more con-ventional circuits. In a highly complex display some defective pixels may be unavoidable and for some applications may be quite acceptable but line defects are very undesirable and so their elimination in the CCT circuit is very significant.

The drive waveforms for displays based on the SLM circuit are shown in Figure 8. The frame time is equal to 2T. In the odd half-frames the refer-ence voltage (V_{REF}) is zero and the data voltage (V_{DATA}) is equal to zero for off-pixels and to V_{ON} for on-pixels. In the even half-frames the logic levels are reversed, ie V_{DATA} = 0 corresponds to on-pixels and V_{DATA} = V_{ON} to off-pixels. In this way the voltage polarity on the liquid crystal pixel (V_{LC}) is changed every half-frame (T). For the CCT circuit a modification of the drive waveforms is necessary. First we observe that, since two half-pixels must be charged in series, V_{DATA} needs to be equal to $2V_{LC}$. If the scheme of Figure 8, without further modification is used, the voltage excursion between source and

drain would be as high as $4V_{LC}$ (ie about 20V for twisted nematic display). This would result in high off-currents and, eventually, breakdown of the device. of the device. For this reason we have used the drive wafeforms of Figure 9. In this case the half-frame time T is a multiple of the refresh time T_r. In the last refresh time before the end of each half-frame all pixels are switched off. In the rest of the frame time the display is addressed in the normal way. By using this scheme the source drain voltage excursion is limited to $2V_{LC}$, which is perfectly acceptable.

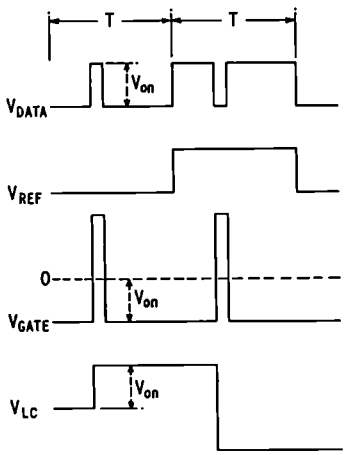


Fig.8 Drive waveforms for SLM circuit

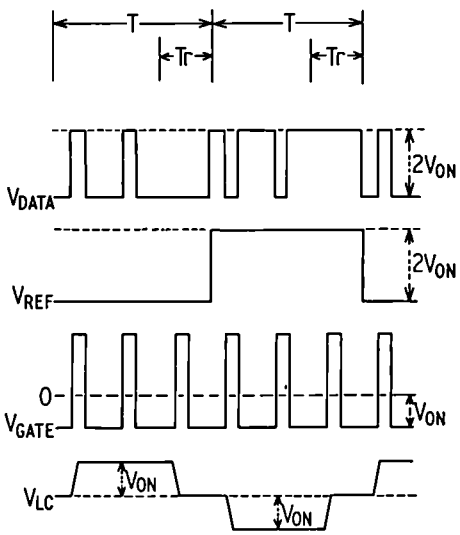


Fig. 9. Drive waveforms for CCT circuit

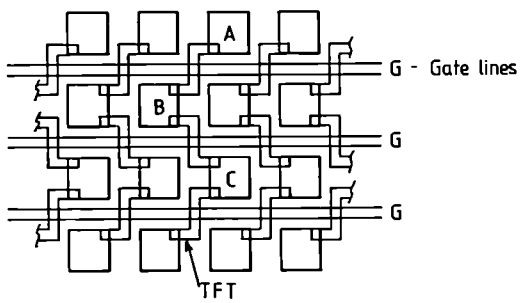
We wish, finally to point out that this scheme used here to achieve polarity reversal on the liquid crystal has a considerable advantage in terms of drive circuitry requirements over the standard method. In the latter the data voltage voltage is changed from positive to negative values on alternate half-frames while the reference electrode is always at V = 0. This would require drivers with a maximum rating V_{MAX} equal to $2V_{LC}$ for the SLM circuit and $4V_{LC}$ for the CCT circuit, whereas for our drive waveforms $V_{MAX} = V_{LC}$ in the SLM case and $V_{MAX} = 2V_{LC}$ for the CCT case.

The successful implementation of the CCT circuit depends on low parasitic capacitances between TFT gate and the source and drain terminals. The self-aligned structure used for polysilicon TFTs make them very suitable as it minimises these capacitances, and the relatively high carrier mobility of poly-silicon means that the required on-currents can be achieved in TFTs with minimum width which further reduces the parasitic capacitances of the source and drain of the device. Amorphous silicon TFTs are not usually fabricated with a self aligned process and so suffer from larger parasitic capacitances which will impair the effectiveness of the circuit. They do, however, have one advantage when considered for this circuit configuration. Each pixel is now split into two capactives connected in series so the pixel capacitance is reduced by a factor of four. This effect reduces the required TFT ON and OFF currents in proportion. This favours amorphous silicon TFTs which cannot supply such

large currents due to the low carrier mobility of the material whilst the difficulties in achieving the lower OFF current requirements with polysilicon will limit display complexity with present devices. Nevertheless demonstration displays using polysilicon TFTs in the CCT circuit have been successfully fabricated and are reported in section 5. Further improvements in polysilicon TFT performance are also anticipated as mentioned in Section 1.

In the CCT circuit each pixel is split into two half pixels to which the same voltage is applied. The gap between pixels , typically 20μm wide, is wasteful in terms of real estate and limits the achievable aperture ratio (AR), i.e. the ratio between active area and total display area. The problems become increasingly serious for high resolution displays. We have now invented a new circuit which allows us to apply a different voltage to each half-pixel, thereby overcoming the above problem. This is obtained without any complication in the processing or the addressing of the display. The circuit is shown in Fig.10.

A) ACTIVE MATRIX BACKPLANE



B) TOP PLATE ELECTRODE ARRANGEMENT

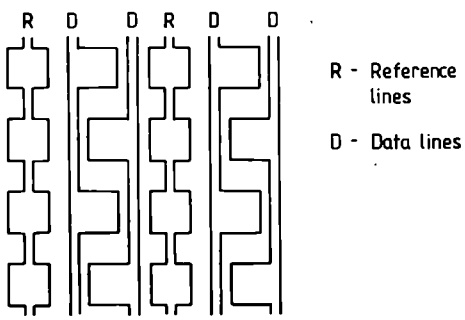


Figure 10: New polysilicon TFT circuit design

It employs the novel concept that if the source and drain of the transistor are fabricated on opposite sides of the gate bus line they can then be connected to pixels belonging to adjacent columns. When the nth row of gates is addressed, the same voltage V_A is applied to pixels A and B. When the (n + 1) th row is addressed the voltage V_B is applied to pixels B and C and so on. In this way a different voltage can be applied to pixels connected to the same transistors.

The scheme has an additional advantage in that it has a built in redundancy. If one TFT is faulty and is removed, e.g. by laser trimming, the pixels will present the information of the next connected pixel which will partially hide the visual effect of the fault. Alternatively, one could arrange for the display to be scanned from top to bottom and bottom to top in successive frames which would additionally average the effect of the faulty pixel.

A further circuit has been devised (Fig.11) which makes it easier to drive colour displays having a triangular RGB arrangement. A triangular RGB arrangement is supposed to give the best visual rendition for TV display. It is evident from Fig.11 that each colour is associated with one specific data bus irrespective of the field, which simplifies the driving of the display when each frame results from interlacing two fields as in the NTSC TV system.

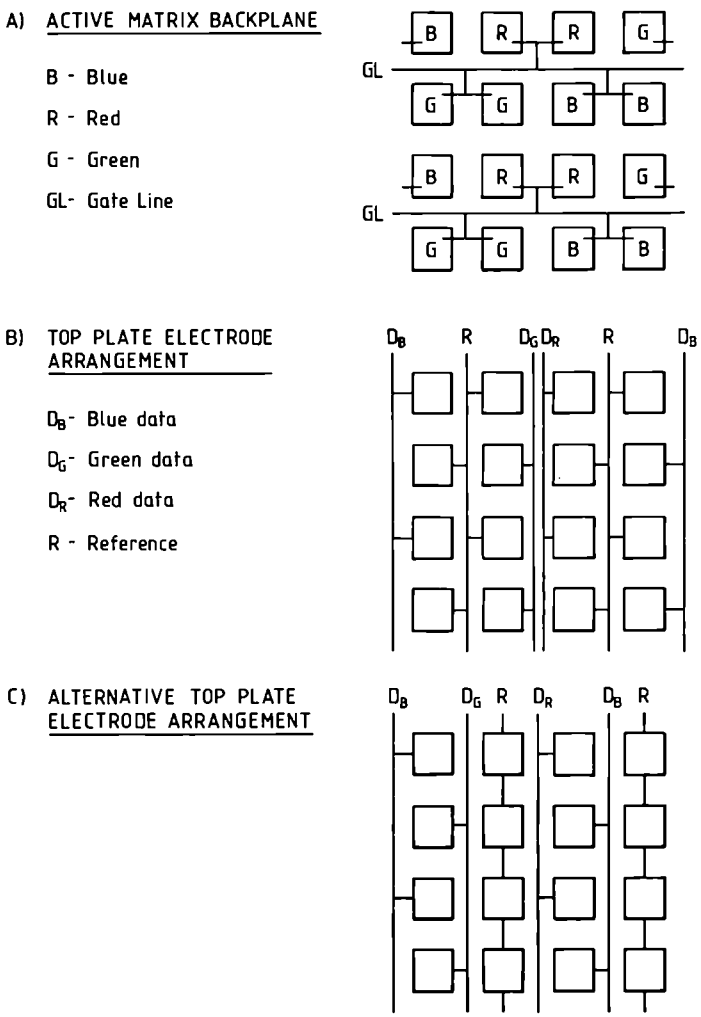


Figure 11: Pixel arrangement for colour display.

4. PANEL FABRICATION

4.1 Cell Technology

Plastic beads from Nippon Skokubay have been used as spacers in the liquid crystal cells.
To deposit the beads on the glass plate two different methods have been developed at Thomson CSF:

- dry spray : the beads are blown by N_2 jet

- wet spray : the beads are dispersed in freon and the solution is sprayed on glass plates.

The second method gives very good results in terms of dispersion homogeneity. 50 to 100 beads per mm^2 must be deposited on the plates to obtain the require LC cell thickness, (or 10-50 glass fibres). The homogeneity and accuracy of the thickness are better with plastic beads than with glass fibres. The influence of the type of spacers (fibres or beads) on the number of row-column short-circuits has not been clearly determined to favour one type of spacer or another.

At AEG the processes of the existing LCD production were carefully analyzed to meet the special requirements for TFT displays. Several new LC materials with and without dichroic dyes were tested to assess contrast ratio and angular distribution. An efficient diffusion blocking layer made of CVD SiO_2:P has been developed in view of using cheap soda-lime glass substrates By using a computer programme, the reflection of light on the multilayer system was minimised for high quality displays.

At GEC during the first six months of this project the general problems associated with achieving uniform spacing and good surface alignment were overcome. However, the novel cell geometries advised as a result of Task 1, such as the capacitively coupled transistor (CCT) structure have led to greater demands on the processes uses to assemble the final liquid crystal display.

In particular, it is essential that the electrode patterns on the two substrates are registered to within $\pm 10\mu m$. While it is fairly straightforward to align the electrode patterns to the required tolerance, it was found that mis-registration occurred as a result of the subsequent sealing process. The latter takes place in two stages:-

(i) Tacking; to fix the location of the substrates. This achieved by placing the aligned substrates held together by 4 clips in an oven at 100°C for 1 minute.

(ii) Assembly of the display in a heat press operated at 180°C, 50 p.s.i. for 5 minutes.

It was found that the majority of the movement and hence misregistration occurred at the tacking stage. This is probably due to the difference in coefficients of the thermal expansion of the two substrates. This problem was alleviated to some extent by using a UV curable adhesive instead of the thermal tacking process.

The Capacitively Coupled Transistor (CCT) design, unlike the conventional circuit, requires patterning of the indium-tin-oxide (ITO) on the top plate of the cell. Carefully controlled photolithography is required. Although the

finished top plates are rectangular, the processing has been carried out on square substrates in order to achieve better uniformity of photoresist. (An alternative is to use a spinner chuck recessed to hold the substrate.) Adhesion of the photoresist to the ITO is obviously crucial and can be a problem; however, we have found that with appropriate baking, satisfactory results may be obtained both with negative (Countdown MRBO) and positive (Shipley 111/S) resists. An alternative solution is to use an additive process; we have successfully used an image reversal resist which lifts off easily in acetone, although it is less sensitive than the Countdown resist, necessitating longer exposure times. With development this problem may be overcome and the scope for increased resolution offered by this technique will be a distinct advantage.

4.2 Interconnections

The interconnections between the driving electronics and the active matrix display are a common problem as the display size and its complexity increase. An A4 flat panel display will require about 1500-2000 connections on a pitch of 0,75 -0,5mm to be made to the drive circuit. There are three basic techniques available:

1) Use of conductive elastomer/glue
2) Chip-on-glass.
3) Fully integrated drivers.

For several reasons the work has concentrated on technique (1); first of all it gives the possibility of combining a tested display with a tested drive circuit, and thereby achieve a relatively high overall yield; and secondly the technique requires no further processing (e.g. metallization) of the glass substrate.

For applications that allow extra glass space (e.g. a margin round the active area of the display) technique (2) is a solution with nearly the same characteristics, especially when the TAB-technique that gives the possibility of desoldering of IC's is used.

The study included oriented conducting film connectors between an LCD and a driving circuit on a flexible printed circuit board and the use of tape automated bonded IC's (TAB). These methods give the following advantages:

° Simple and light mechanical construction.
° Easy backlighting option
° Minimum space requirements
° Relatively high yield and reliability

To study the connection technique described above, two samples of ordinary LCDs were used, which give the same basic connection problem as for active matrix displays.

In the first example, a static LCD was connected to a flexible PCB with 4 TAB-drivers. There are two connection sections (two edges) of 47mm each and the pitch is 0,75mm. The oriented conductor film connector was shown to be reliable after the processing parameters were correctly defined (temperature, pressure and time).

The second prototype was a multiplexed LCD (31cm x 5cm), which was connected to 12 LCD-drivers. There were 8 connection sections each of 62mm with a pitch of 1mm. Although the pitch is higher, this prototype was quite close to the required solution for an A4-display with connectors on the 4 edges. Also, in

this case it was proved that the process was usable and showed appropriate reliability.

The conductive film used for the connections was a product from a company in Japan. It is a polymer film, which exhibits orientation in its conductivity and has three functions. Adhesion between flexible PC and glass, conduction from FPC electrodes to glass-electrodes and isolation horizontally between the electrode pairs.

The oriented conductor film connector consists of carbon fibres of 5-100mm length dispersed in a film with an adhesive (30mm thick) as a binder. The film is delivered in strips and during the bonding is sandwiched between the two electrode patterns to be connected.

The sandwich is heated and pressed for the prescribed time. The binder adhesive then performs both bonding and insulating roles, while the carbon fibres within the binder are brought close to the electrodes and provide conductivity.

The connector construction is mechanically strong in itself, but for demanding applications it might be necessary to add an extra layer of glue or a metal feather to strengthen the assembly. A semiautomatic bonding machine was constructed to perform the heat-press process. In the following the verified optimum conditions are specified. The FPC connectors are gold-plated.

Temperature (at film)	:	146°C
Temperature (at tool)	:	169°C
Pressure	:	78 bar
Bonding time	:	12 seconds

5. COLOUR DISPLAYS

A colour display is achieved by adding to a B/W panel, in front of each pixel a colour filter and,on the back-side, a white light source. Since the pixel pitch is not much larger than the cell glass thickness, the filters must lie inside the cell to avoid parallax. A colour active matrix liquid crystal display will then consist of two plates, a first with the TFT matrix and a second one with colour filters and ITO electrode.

Two filter processes have been developed in the present ESPRIT programme. The Thomson process was derived from the one previously used for solid state sensors

It was based on dyes locally diffused in gelatin then passivated and coated with ITO. Special care has been taken to yield the process compatible with the LC cell fabrication. Black matrix fabrication has been studied to increase colour saturation and shield the TFT from light.
CNET used pigmented photoresist for RGB filters and metallic film for Black Matrix. Filters were patterned as normal photoresists but a short curing took place after each colour. Black matrix was first realised on the glass substrate. It was deposited at room temperature on the base filters and passivated by the orienting polyimide layer. The total thickness of colour filters is about one micrometer.

A checking of the AMLCD compatability has been performed, particularly the non-pollution of the LC material by the filters. On CNET samples, for example the electrical time constant "RC" (the voltage decay across the pixel on the OFF state is the relevance of this parameter) was not affected by the presence of the filters and in any case was found to be lower than 80 ms.

The colorimetric measurements were also performed and results are shown on Fig. 12.

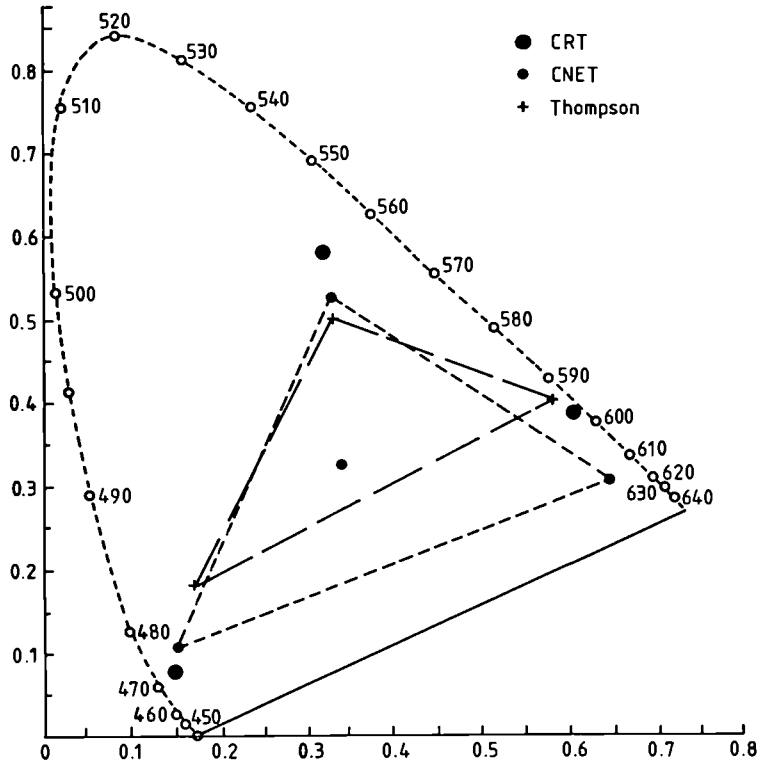
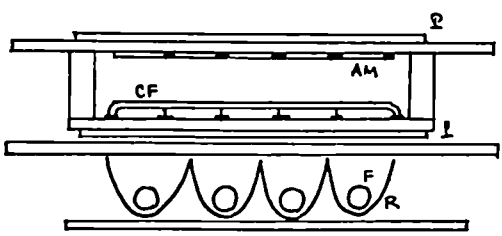


Figure 12: Colorimetric measurements on the filters.

The colour cell fabrication was not so different from the B/W panels. Care has been taken to avoid over-heating of the filters. CNET filters can be heated up to 200°C. The structure of a colour AMLCD is shown in Fig.13.



AM : Active matrix (TFT - aSi:H)
CF : Colour filters
(with black matrix)
P : Polarizers
D : Weak diffuser
F : Fluorescent tubes
R : Metallic reflector.

Figure 13: Colour AMLCD structure

A typical result achieved by CNET is shown in Fig 14, which displays a broadcasted CTV picture. The panel characteristics were as follows:

Dots: 320 x 320 - TFT aSiH - CNET process

Filters RGB vertical stripes (3 x 107)

Pitch : 250 µm

Active area : 80 x 80 mm

Filter process : Pigments in photoresist

ITO on filters : R = 700Ω/sq.

LC thickness ~ 8 µm

Fig.14: A typical displayed CTV picture on the CNET panel

A contrast ratio of more than 30:1 was measured at normal incidence for achromatic pictures (R=G=B). No streaking was observed. Flickering appears at the limit of the angle of view. It can be suppressed by alternating the video signal onto the odd and even columns. A comfortable viewability is possible over a wide angle (at least ± 40° in the horizontal plane).

6. DISPLAY CHARACTERISTICS

The characteristics of the Thomson CSF colour display presented at the Esprit Technical Week and developed under the 833 programme are as follows:

- inverted staggered amorphous silicon TFTs with silicon nitride (Si_3N_4) as gate dielectric and channel defined by W/L = 20/10 µm.

- pixel size 210 x 210 µm

- pixel pitch 250 x 250 µm
- width of the access lines (rows and columns), 20µm.
- number of lines : 256 rows : 320 columns
- active area : 64 x80mm²
- connection pitch (interlaced connections): 500µm
- overall size : 98 x 114 mm²
- liquid crystal cell thickness : 8µm
- spacers : calibrated glass fibres
- polarizers : 99.9% polarizing efficiency
- colour filters : red, green, blue, along one diagonal deposited over the counter electrode, covered with a passivation layer and a conductive ITO electrode coating.

The 256 x 320 display developed by Thomson CSF is shown in figure 15:

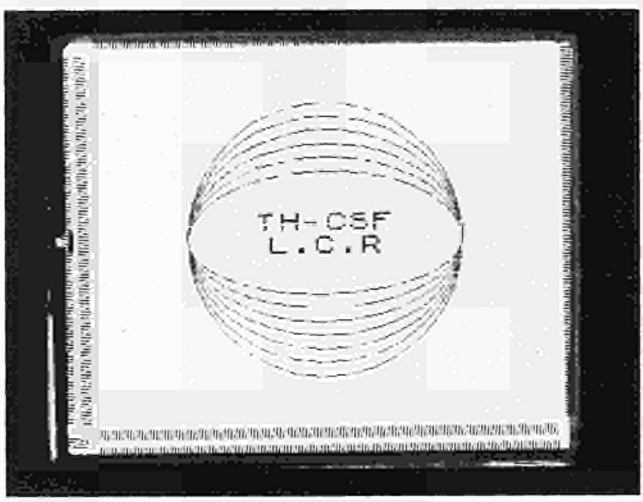


Figure 15: 256 x 320 pixel display.

The drivers are connected at the periphery of the display for rows: 4 drivers with 64 connections (maximum voltage output 26 volts); for columns (video): 4 drivers with 80 connections (maximum voltage 26 volts); The image supplied to the display comes from a Thomson 107/70 microcomputer and is fed to a memory which allows addressing at a frequency of 100Hz (twice the video frequency). Backlighting is provided by spectrally matched fluorescent tubes (15 watts). The luminance achieved in the white is 180 col/m² and the contrast is 20:1.

The influence of TFT driving conditions on contrast ratio was measured on a panel with 28x28 pixels and standard TN LC material. Pixel area was 1mm². TFTs were produced with SiO_x and a-Si. Both deposited by PECVD-techniques.

Contrast ratio was measured by illuminating only one pixel with monochromatic light.

In figure 16 the influence of gate-pulse voltage V_G on contrast ratio is illustrated for optical wavelength λ - 550nm. At conditions:

frame time	10 ms
gate-pulse time	50 µs
duty cycle ratio	1:200
gate voltage for TFT off-state	-10V
source voltage	10V

saturation contrast ratio was obtained for gate-pulse voltage V_G > 15 V.

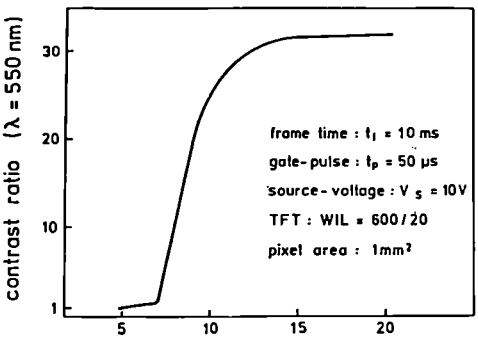


Fig.16: Influence of gate pulse voltage on contrast ratio

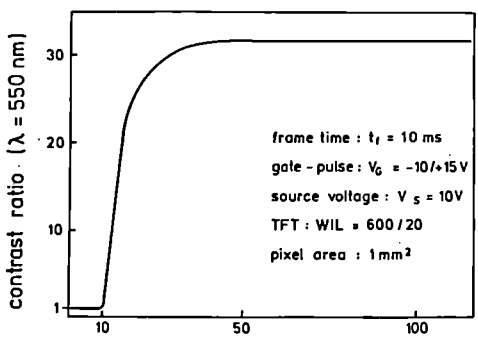


Fig.17: Influence of gate pulse length on contrast ratio

In Figure 17 the influence of gate-pulse length t_p on contrast ratio is shown for λ = 550nm. at conditions:

frame time	10 ms
gate voltage for TFT off-state	-10V
gate-pulse voltage	15V
source voltage	10V

saturation contrast ratio was reached for t_p > 40µs.

The same behaviour was obtained at two other optical wavelengths, for which, however, different saturation contrast ratios of 22 at λ = 450 nm and of 11 at λ = 650 nm were measured.

An estimation of the theoritically-required gate-pulse length t_p was carried out. The basic premises according to former measurements were as follows:

- TFT on current, $I_{on} = 2 \times 10^{-6}$ A at V_G = 15 V and V_{DS} = 10 V
- rise time, t_r < 3 µm for I_{on} at channel length L = 20 µm
- for saturation contrast the LC pixel capacitance has to be charged up to ± 4V

With these premises the estimation yields t_p = 30 µs, which is in good accord ance with the experimental result reported above. However, for smaller pixel areas at higher resolution shorter gate-pulse lengths t_p are sufficient to charge the reduced LC pixel capacitance.

Polysilicon TFT matrix addressed LCDs of up to 224x208 pixels based on the CCT circuit configuration described in section 2 have been fabricated. Figure 18. is a photograph of the 224x208 pixel display driver using the waveforms described in section 2.

Fig.18: 224 x 208 polysilicon display

Typical display parameters are as follows:-

Circuit Configuration	CCT Circuit
TFT	W = 14µm L = 14µ
Pixel count	224x208
Pixel pitch	400µm both ways
LC	Twisted nematic
V_{GATE}	-5V +18V
V_{DATA}	10V
T_r	5 ms
T	15 ms
Contrast	20:1

6. USER INTERACTION

The development of office machines and interactive services on the Telecom network will incorporate the design of new terminals able to send and receive hand-written messages. An attractive terminal will consists of a Visual-Display Unit for any pictures lashed to a data-input device for hand-drawing messages,

and connected to the telephone network by means of a signal processing unit. From the point of view of the man-machine dialogue the best set up would be a flat board to display and write on, as similar to a sheet of paper as possible.

CNET have fabricated a prototype of an interactive terminal, associating an horizontal Active-Matrix Liquid-Crystal-Display with a transparent tablet on the top. The user can write on it by means of an electronic pencil. This prototype operated only in the local mode, and was used for performance valuation.

The first results of viewability, writing speed and linearity are reported here. Figure 19 shows a schematic arrangement of the hand-writing terminal.

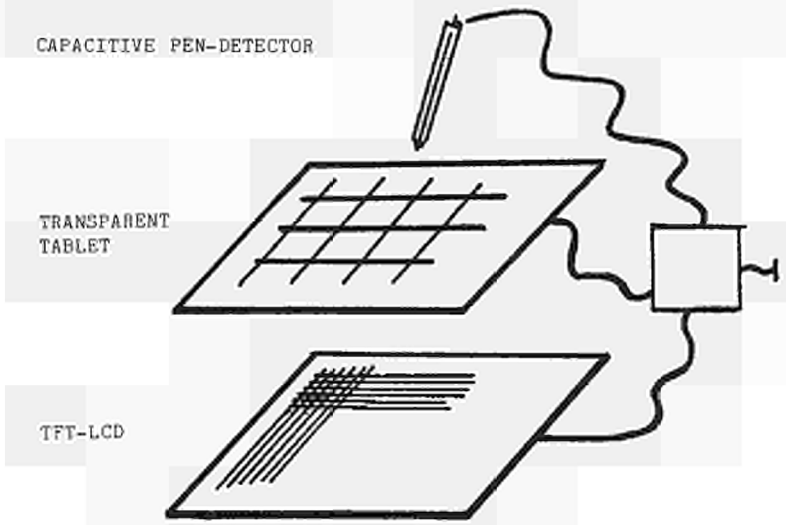


Fig. 19: Interactive terminal

The different pieces of the hand-writing board are stacked as follows: the light guide with fluorescent tube, the LCD with polarizer and the transparent tablet on the top. The total thickness is less than 15mm.

The principle of this data-input device is an electrostatic coupling between an x-y array of linear conductors powered by drivers, and a capacitive pencil-receptor. The x-y electrodes were successively driven by phase-shift-registers (100 KHz clock frequency), the pen circuit transformed the analog to digital signal and converted the spatial into time delay detected by digital counters. This circuit has been designed to get a spatial accuracy of 0.4mm with an electrode pitch of 3,2mm.

The flat panel display has been already described elsewhere.(3) This TFT-addressed LCD had an active area of 100 x 128 mm^2 and a pitch of 0.4mm.

It was manufactured according to the "CNET 2 step process" (1). Columns and pixels were in ITO and lines in Al/SiN/aSi:H. The panel operated between parallel polarizers.

Figure 20 shows an example of written message. The writing size must be adapted to the panel resolution to avoid rough letters.

Writing speed seems satisfactory for normal use with no visible delay between pen motion and display. Linearity was satisfactory for writing and drawing.

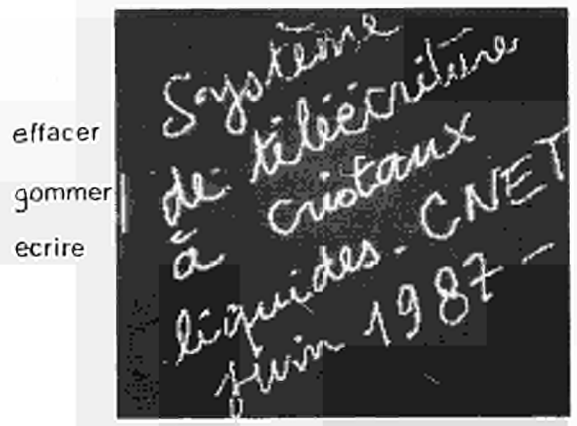


Fig. 20: Example of written message.

CONCLUSION

In this paper we have reported technical progress in a number of important areas related to the fabrication of large-area high-information-content liquid crystal displays using silicon TFT active matrix addressing.

Following the excellent results obtained in the opening phase of the programme on both a-Si and poly-Si TFTs, the size and the complexity of the matrix-addressed displays produced have been increased considerably, and attractive demonstrators with a low level of defects have been fabricated. A new circuit has been demonstrated which removes the possibility of line defects occuring due to transistor faults by isolating the transistor from the rest of the circuit using the liquid crystal. This also has the effect of increasing yield by reducing the complexity of the backplane by removing the data lines to the top plate.

The work has progressed far beyond fabrication of the active matrix alone with the incorporation of both colour and user-interaction, features which are in great demand in the display market.

The potential worldwide market for such displays is enormous since applications for these displays include domestic TV sets and computer and telecommunication terminals.

The size of the world market for LCDs in 1986 was 780 million units with a value of 870 million dollars and it is predicted to rise to 1,200 million units with a value of 1,600 million dollars in 1990. The large area high resolution market is represented by CRTs for which the figures are 96 million units and 4900 million dollars in 1986 rising to 120 million units and 6000 million dollars in 1990.

Flat panel displays will compete with CRTs for this market, but also, because of their low voltage operation and low power consumption and due to the relative ease of implementing user-interaction there are openings for many new

applications which were not possible with CRTs. It is therefore hardly surprising that there is a large worldwide R & D effort directed towards the fabrication of such a display and competition in this field is very keen. At least four Japanese companies have already demonstrated small laboratory versions of full colour TVs, and two products are already on the market (Seiko-Epson and Matsushita-Panesonic). At least eight Japanese companies are involved in AM-LCDs in one way or another. The effort in the US has been neither so strong nor so successful but is now increasing with both IBM and GE having put together powerful teams.

The feasibility of producing such displays up to A4 size has been demonstrated by Seiko Instruments using amorphous silicon technology, and a number of Japanese companies have production equipment capable of handling this size of substrate. In Europe, however, no such equipment was available, and the large area projection aligner and large area amorphous silicon deposition equipment which were developed under this contract have gone some way to redress the balance in this respect but in order to compete with the Japanese companies in production the development of processing techniques and large area equipment must be a priority in any future work plan. Over the next few months, the prototypes which have been developed will be evaluated and the possibilities for other key processes will be investigated.

Future work will also include further evaluation of the present displays and assessment of defects, improvements of display features and preliminary investigations of the use of polysilicon TFTs for integrated drivers.

At present no decision is to be made between the amorphous and polycrystalline silicon approaches. For a-Si the effort will be concentrated on implementing grey-scale and increasing the size of the displays. For polysilicon, deposition equipment for A4 needs to be developed, and alongside this effort the present displays will be improved by increasing the resolution, incorporating colour and implementing grey-scale. An evaluation of the dyed phase change effect for use in this application will be undertaken and a demonstrator fabricated for comparison with standard twisted nematic displays.

The results obtained during this 18-month feasibility study have placed the consortium in a very strong position as far as device understanding and novel ideas are concerned. The effort now needs to be concentrated into exploiting these advantages to develop a competitive product which will provide an attractive addition to the European initiative towards the next generation of office systems and portable computers.

REFERENCES:

1. C. Hilsum et al, ESPRIT '86: Results and achievements. Elsevier Science Publishers B.V. (North Holland) 1986

2. D.B. Meakin et al, to be published

3. F. Morin et al, Japan Display 86, p332

Project No. 491

POLY-SI THIN FILM TRANSISTOR TECHNOLOGIES FOR LIQUID-CRYSTAL DISPLAY DRIVERS

W. Senske, W. Schmolla, J. Diefenbach, G. Blang
AEG-Forschungsinstitut, Goldsteinstrasse 235, D-6000 Frankfurt 71, FRG

B. Loisel, P. Joubert, Y. Chouan
ROC, CNET, Route de Tregastel, 22302 Lannion, France

M. Krammer
CETIA, 150 rue Marcelin-Berthelot, 83088 Toulon, France

The fully integrated TFT based matrix LCD is the ultimate goal for flat panel displays. This paper discusses material and technologies to fabricate poly Si TFTs that have to meet the requirements for driving LCD matrices. With LPCVD, PECVD and e-gun evaporation poly Si TFTs have been fabricated at deposition temperatures as low as 550°C and mobilities between 2 and 35 cm^2/Vs. The potential is indicated to further decrease the deposition temperature even to values where very cheap soda lime glasses can be used. The technologies developed for poly Si will have impetus on other areas of large area electronics.

1. INTRODUCTION

Flat panel displays for text and graphic information are of great importance for products in the information technology industry. During the last years the liquid-crystal (LC) display has evolved to a highly reliable display technology since it is visible in bright light, consumes little power, has a good visibility and a broad angle of view. For replacing the bulky CRT screen the active matrix LC display (AMLCD) is best suited when a large information content is to be displayed. These AMLCDs with TFT addressing of the pixels have attracted considerable industrial interest and are at the threshold of being commercially available. By that, text and graphic displays in colour with high resolution, high contrast and viewing angle become possible by a rather cheap thin film technology on glass substrates. Up to now the majority of AMLCD concepts involves external driver circuits consisting of silicon IC's assembled on PCB's that are connected to the pixel TFT matrix.

For the full advantage of thin film technology for text- and graphic AMLC displays the external electronic drivers must be incorporated on the glass substrate, too. Contrary to pixel TFTs the driver TFTs require a material with high mobility. In general this kind of material needs a high deposition temperature. Up to now this has hampered the fabrication of fully integrated matrices. Few attempts have been devoted to fabricate integrated pixel and driver TFT matrices on a single substrate (1). However, the fabrication process necessitates the use of quartz substrates which are too expensive for large area displays and high volume production.

The objective of this investigation is therefore mainly to develop a low temperature deposition process of a semiconductor material so that cheap glass substrates can be used. For reasons of higher switching speed, as compared to pixel TFT's, these semiconductor films must have markably higher field effect mobility than amorphous silicon which is most frequently used for pixel TFTs.

In this paper we report achievements that have been made concerning the low temperature deposition of polycrystalline Si films. The deposition methods investigated are e-gun evaporation, LPCVD and PECVD. After a brief discussion of the material requirements for integrated bus drivers some results concerning the material fabrication and the poly Si TFT deposited on glass are presented. Finally some studies concerning simulation of TFTs and TFT circuits are discussed.

2. REQUIREMENTS FOR SEMICONDUCTOR MATERIAL

The choice of the deposition method has to consider that in principle substrates up to A4 size can be treated. Although the integrated driver circuits only need a stripe of a few mm width at the edge of the display larger areas of substrates have to be taken into account for reasons of full integration. A considerable factor of cost for the large area displays will be ion implantation if necessary for n^+ contacts. Therefore one objective of the investigations is to avoid this costly process including demand of high annealing temperatures.

The deposition temperature aimed at is closely related to the substrate material that is avalaible. There are two types of low priced glasses

(i) borosilicate glass (e.g. Corning 7059, Hoya NA 40, Schott AF 45) with maximum deposition temperature of about 630°C

(ii) soda lime glass (e.g. window glass) with maximum deposition temperature of about 500°C

The borosilicate glass is considerably more expensive than soda lime glass due to different fabrication processes. For a full integration of LC matrix and drivers it is of great industrial interest to use the cheapest type of glass that is possible.

The realization of seperate large area driver circuits on borosilicate glass may be an intermediate step. However, this solution has to compete with the chip- on-glass technology where the IC is directly soldered onto the edge of the matrix substrate. In both cases the interconnections between driver and pixel matrix are a critical point in view of reliability. This disadvantage is avoided with full integration.

The most important parameters for the driver circuit material are the switching speed and the mobility. Assuming a TFT matrix with 300 columns and 300 lines the clock rate for the column shift register amounts to about 100 nsec provided the AMLCD is driven with 32 Hz frame rate. The switching time of a single TFT has to be lower than this clock rate. The inverter is the simplest stage of a shift register (2). An inverter can be constructed with two transistors. One transistor acts as load resistor. The other is the switching TFT. A rough estimate for the minimum field mobility can be obtained by calculating the current that is necessary to charge the switching transistor of the next inverter stage for given geometries and voltages. The calculation results in a minimum mobility of about 10 cm^2/Vs. This result shows that for integrated TFT drivers the a-Si and the so called micro-crystalline Si (μc-Si) as a mixture of a-Si and small grain Si are not suitable due to their too low mobilities. Thus the polycrystalline Si (poly Si) has evolved as the most favorable candidate.

The values mentioned for clock rate and field effect mobility are only rough estimates. To get more accurate figures and to investigate possible shift register structures the technological work is accompanied by simulation. Thus it is expected that critical material parameters as well as requirements for TFTs concerning switching time and geometry can be identified.

3. POLYSILICON DEPOSITION

For pixel TFTs in AMLCDs the a-Si has presently become the most favorable material. The main reasons are the low deposition temperature which allows the use of cheap soda lime glass and the ease to obtain large on/off ratios as well as low off currents which are prerequisites for large area full colour displays. The recent results concerning poly Si deposition on non-quartz substrates have stimulated the investigations towards a fully integrated matrix (3)-(6). In this paper some results of three different deposition methods will be presented namely of LPCVD, PECVD and e-gun evaporation of poly Si film.

3.1. LPCVD

Recently the LPCVD (low pressure chemical vapour deposition) has attracted considerable interest for poly Si deposition on borosilicate glass (3),(6),(7). However, this deposition is a thermally activated process. Therefore the deposition temperature cannot be lowered beyond a certain limit of substrate temperature which is about 580°C. Below this limiting temperature the deposition rate is no longer acceptable.

In the present investigation, the LPCVD technique has been chosen to establish the TFT technology on borosilicate glass substrate. In the course of this work some improvements of the deposition method have been found.

In previous papers on LPCVD deposition of undoped poly silicon films (8),(9), the transition temperature between amorphous and crystalline silicon as well as the effects of temperature on the size and the preferred orientation of the crystallites have been reported. However, the effects of silane pressure have not been published extensively especially at very low silane pressure. We report results on the structure of the LPCVD poly Si films deposited over a temperature range of 580°C to 700°C and a pressure range between 2×10^{-3} and 10 Torr (6).

The films were deposited on fused quartz substrates for the high temperature range and on borosilicate glasses (Corning 7059, Hoya NA-40, Schott AF-45) for the lower temperature range (580°C - 600°C) by thermal decomposition of silane (SiH_4) diluted in hydrogen (10% SiH_4 - 90% H_2) in a hot-wall reactor.

The main results are:

- All films deposited in the temperature range 580°C - 700°C are crystalline, except the one grown at the lowest temperature (580°C) and the highest pressure (10 Torr). As shown on the Table 1, the crystallite size (measured by x-ray analysis) decreases when the pressure increases especially at low temperature

- The texture of the films is strongly dependant on the silane pressure inside the chamber (Fig. 1).

- The surface roughness of the films increases with increasing pressure of the silane (Fig. 2) at 700°C.

Table 1
Crystallite size determined from (220) x-ray diffraction line on 1 µm-thick poly Si film deposited with a 40 sccm gas flow (10 sccm at 2.10^{-3} Torr) at five different values of gas pressures of gas (10 % SiH_4 - 90 % H_2).

Temperature (°C)	XRD Crystallite size (nm)				
	0.002 Torr	0.02 Torr	0.1 Torr	1 Torr	10 Torr
580	-	82	60	39	amorphous
625	120	110	116	92	62
700	102	138	136	105	-

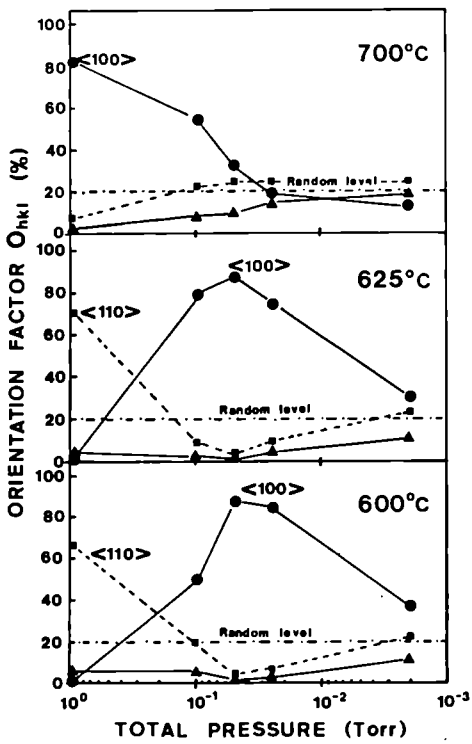


FIGURE 1
Orientation factors (▲: O_{111}, ■ : O_{220} and ●: O_{400}) measured by X-ray diffraction on poly-Si films deposited at various temperatures and gas (10 % SiH_4 / 90 % H_2) pressures.

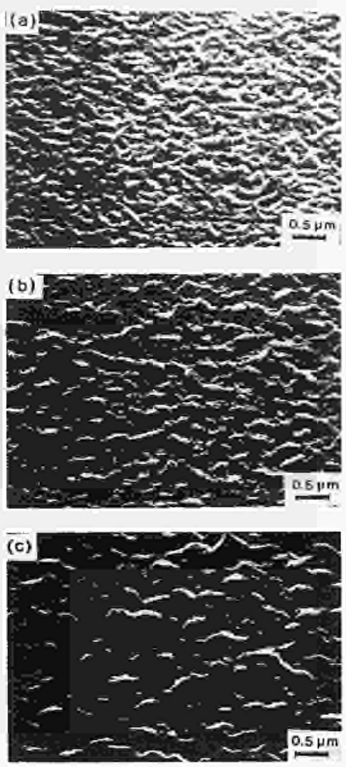


FIGURE 2
Scanning electron micrographs of the surface roughness of LPCVD polysilicon film deposited at 700°C with three different gas pressures: (a) 2.5 x 10^{-3} Torr (randomly oriented grains), (b) 5 x 10^{-2} Torr (weakly oriented $< 100 >$), (c) 0.1 Torr (strongly oriented $< 100 >$).

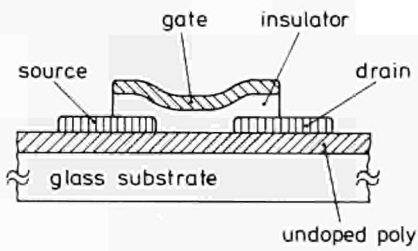


FIGURE 3
Schematic structure of poly Si TFT with normal staggered structure

TFTs have been fabricated with the normal staggered structure (Fig. 3) by using Al Si alloy contacts and a Si_3N_4 gate insulator. Typical output characteristics of these transistors are given in Fig. 4 and 5. The transistors behave like an n-channel enhancement TFT. For W/L = 1,5 the drain current for V_{DS} = 20 V and V_{GS} = 20 V is higher than 20 µA. At low V_{DS} the drain current shows some crowding. A field effect mobility of 1.2 cm^2/Vs and a threshold voltage of 3 V have been deduced. The subthreshold behaviour of the TFT shows some hysteresis (Fig. 5). The off-current of V_{GS} = -10V is less than 10^{-8} A.

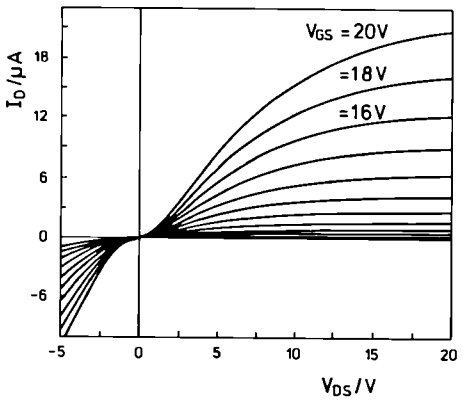


FIGURE 4
Output characteristics of a TFT fabricated with LPCVD poly Si at 580°C. V_{GS} varies from -10V up to 20V in steps of 2 V.

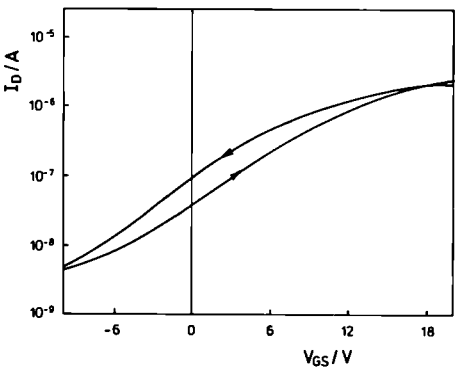


FIGURE 5
Typical transfer characteristic, V_{DS} is set at 2 V, LPCVD poly Si at 580°C.

3.2. PECVD

By using PECVD the decomposition of silane is plasma assisted. Therefore lower deposition temperatures than those obtained with LPCVD have been expected.

The undoped polysilicon films were deposited at 580°C on borosilicate glass by glow-discharge decomposition of silane diluted in hydrogen (10 % SiH_4/90 % H_2). The PECVD system has a triode structure. The third electrode, a mesh, is located above the substrate and is grounded. Further details have been published recently (10). The TFTs have been fabricated with the technology established with LPCVD poly Si films. The output characteristics of the transistors are given in Figs. 6 and 7. For V_{DS} = 20 V and V_{GS} = 22 V the current I_{DS} is higher than 100 µA, but at low V_{DS} again some crowding is observed indicating some contact resistance. A field effect mobility of 35 cm^2/Vs and a threshold voltage of 5.7 V were deduced in the usual way (11). I_{on}/I_{off} ratios up to 10^4 were obtained. For negative V_{GS} nearly no increase in drain current is observed.

The PECVD involves a variety of process parameters. Especially the deposition temperature needs a tight control to obtain poly silicon films with good reproducibility.

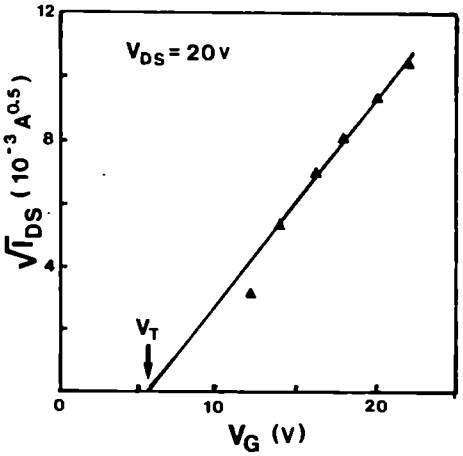


FIGURE 6
Square root of drain current vs. gate voltage measured in saturation region with V_{DS} = 20V for PECVD poly Si TFT at 580°C.

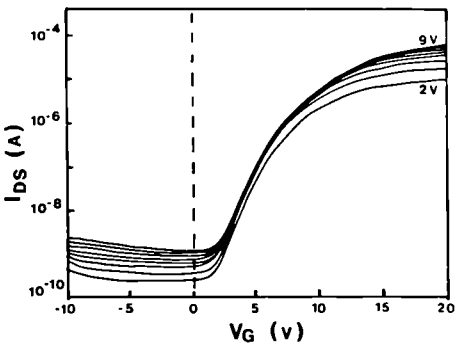


FIGURE 7
Typical transfer characteristics, V_{DS} varies from 2 to 9 V in steps of 1 V for PECVD poly Si TFT at 580°C.

3.3. E-gun evaporation

Recently the electron-gun (e-gun) evaporation technique has been demonstrated to be a very attractive method to fabricate poly-Si high mobility TFTs (4,5).

The poly Si has been deposited at 550°C (4) on borosilicate glass. TFTs have been fabricated by using ion implanted source-drain contacts. For n-channel types field effect mobilities between 13 and 31 cm^2/Vs have been obtained. For p-channel types the mobilities were between 9 and 17 cm^2/Vs without hydrogen passivation and for borosilicate glass covered with CVD SiO_2. For poly Si evaporated on borosilicate glass at 500°C mobilities up to 40 cm^2/Vs at maximum are reported (5) using implanted contacts.

We fabricated thin undoped poly Si films by e-gun evaporation on different substrates and at different substrate temperatures and characterized them by Hall-mobility measurements to get information about the film quality before a complete TFT fabrication process is performed. The results of four selected samples are given in Table 2 below.

Table 2
Hall-mobility for holes in poly Si evaporated from slightly p-type doped high resistivity Si slugs.

substrate material	layer on the substrate	deposition temperature T/°C	Hall-mobility $\mu_H/(cm^2/Vs)$
Borosilicate	no	510	6
quartz	no	500	7
borosilicate	sputtered SiO_2	500	9
soda-lime	sputtered SiO_2	400	2

The Hall-mobility of Table 2 is given for holes in poly-Si bulk material. With sputtered SiO_2 covering the highest mobility amounts to about 9 cm^2/Vs. In conclusion, because of the high mobility in poly Si films on a substrate with SiO_2 covering, we are free of having to use a special substrate material for establishing high mobility values. The covering of the substrate with sputtered SiO_2 is a low cost process and therefore it represents an important factor to increase the mobility.

The Hall-mobility for soda-lime glass with SiO_2 and a substrate temperature of 400°C is 1.8 cm^2/Vs. We have indications that this value can be increased. This would enable us to apply this poly Si technology to high mobility TFTs on low cost soda-lime glass.

Good injection contacts are necessary for the source and drain contacts of a TFT. The ion implantation technique used by other groups (3,4) is not a low cost process for large areas. In addition this technique requires an annealing step at temperatures above 500°C. For these reasons we investigated the doping of poly Si by simultaneous evaporation of Si and Sb from a separate source. Sb doped poly Si films deposited at 500°C substrate temperature have a resistivity of 0.1 cm and an electron Hall- mobility of 2 cm^2/Vs. Doping by evaporation is a low cost process for large areas. This method has a high deposition rate of 1 nm/s for doped films. This method allows to fabricate source-drain contacts for TFTs at process temperatures below 500°C without using the ion implantation technique. Fig. 8 shows the deposition rate of Sb versus the Sb source temperature. Fig. 9 shows the electron concentration of n^+ doped poly Si versus the Sb evaporation rate used for our experiments.

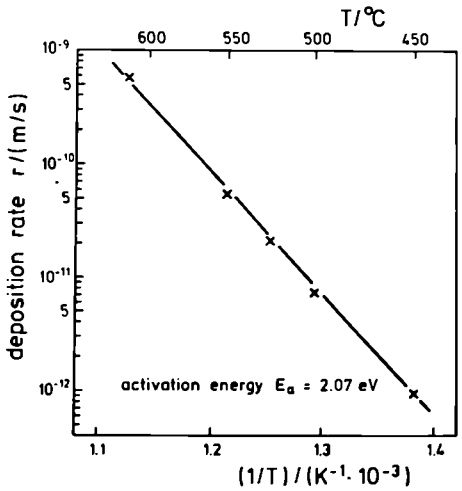


FIGURE 8
Deposition rate of a Sb source for the deposition of Sb onto a cold substrate.

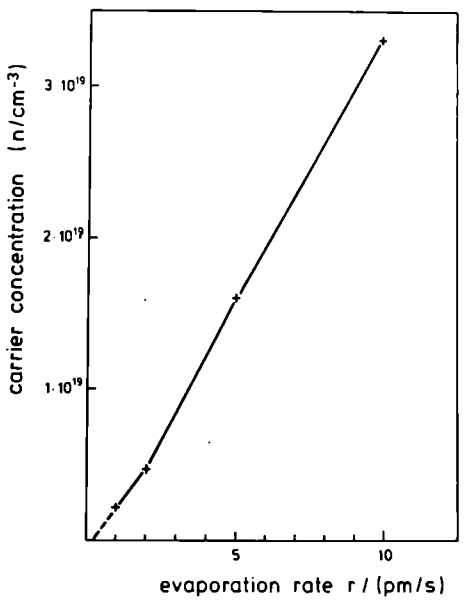


FIGURE 9
Electron concentration in Sb doped poly Si versus Sb evaporation rate for a constant Si deposition rate of 1nm/2, substrate temperature 500°C.

In order to test the application of Sb doped evaporated poly Si and undoped poly Si we fabricated a complete TFT. The structure for the TFT is the same as indicated in Fig. 3. The gate insulator is SiO_2 deposited by APCVD (atmospheric pressure CVD) on borosilicate glass. The highest process temperature was 550°C for the poly Si. The TFT with a gate length of 10 µm has a field-effect mobility of 2.3 cm²/Vs for electrons (Fig. 10). This value has been obtained without any process optimization like substrate or interface treatment. Therefore considerably higher values are to be expected. The key technique is the gate-insulator formation which has been also indicated in previous papers (4), (5).

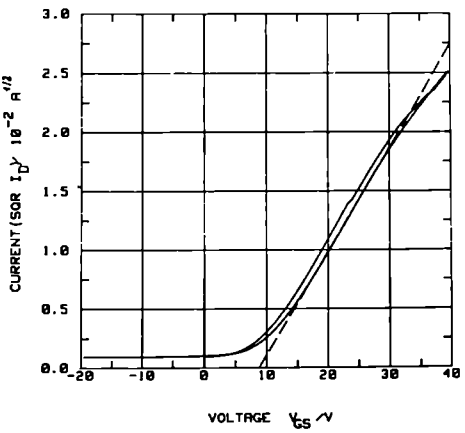


FIGURE 10
Square root of drain current vs. gate voltage with V_{DS} = 30 V for e-gun evaporated poly Si TFT.

4. SIMULATION STUDIES

The objective of this task is to identify critical parameters for the fabrication of TFT circuits like e.g. tolerable overlap capacitances or limits for mobility. Further the suitable structures of the TFT gate and source drivers have to be found.

The program used in these studies is the SPICE model which is a standard program for CMOS simulation with monocrystalline Si ICs. The equivalent circuit for the transistor implanted in the SPICE model had to be replaced by a special circuit which takes into a account the pecularities of a TFT e.g. the insulting substrate. A further difference of TFTs is the increase of drain current with negative V_{DS}. This represents a considerable difficulty for the simulation of dynamic behaviour when using previously published data (4). However, the results presented in this paper indicate that TFTs with considerably less pronounced current increase for negative V_{GS} can be fabricated. Thus the simulation is simplified.

With the SPICE model adapted to the TFT needs different types of shift registers have been investigated. It was found that the ratioed, Fischer type (12) register has the highest frequency response and is less dependant on leakage current of each transistor.

5. DISCUSSION AND CONCLUSION

The requirements for driver TFTs as discussed in section 2 are different from those for pixel TFTs the ESPRIT project 833 has delt with. For pixel TFTs the mobility and the on current are of minor importance whereas the off-current and a low deposition temperature (i.e. a cheap substrate for a large area display) are of great importance. The influence of the threshold voltage V_{th} has to be clarified for driver TFTs. However, for fast and low voltage level circuits this value will certainly be of some importance contrary to pixel TFTs where some variation of V_{th} can be tolerated. For driver circuits the lay out is of major concern e.g. the reduction of stray capacitance or the general structure of the circuit. These considerations are not necessary for pixel TFTs. Thus the project 491 is an important complement to the ESPRIT project 833. More over these investigations lead to technologies that can be used in driving other large area electronics e.g. photoconductive scanners or arrays.

The values mentioned in section 2 are based on rough calculations and do not consider parameters like V_{th} hysteresis or velocity of charging the TFT channel. More accurate simulations are certainly necessary. However, even with smaller values for the mobility a certain degree of integration is possible by using the multiplexing scheme.

In the literature first attempts for fully integrated LCD matrix have been published (1) for small 1,5" displays by using expensive quartz substrates. The transition to large area e.g. A5 or even A4 displays seriously increases the need for high mobility TFTs. Recently values for poly Si TFTs have been published (13) (14) with μ_{FE} 100 cm^2/Vs. Thus a multistage oscillator could be built with 1,9 ns delay time per stage. However, these TFTs have been fabricated with temperatures up to 1000°C by using ion implantation for the source-drain contacts.

An interesting method for low deposition temperatures seems to be the HRCVD (hydrogen radical CVD) (15). By separating the location of the plasma assisted silane decompostion and the film deposition doped poly Si films could be deposited at temperatures as low as 300°C with good Hall mobilities. This method may indicate the potential for the PECVD which is in the present form difficult to reproduce.

In conclusion all three methods investigated in this first phase of the project could be used to fabricate poly Si films on borosilicate substrates. Field effect mobilities between 2 and 35 cm^2/Vs have been obtained. One method, the e-gun evaporation, shows some potential to even decrease the deposition temperature to values where the very cheap soda lime glass can be used. Further work is necessary to improve the reproducibility as well as the TFT characteristics. The simulation has to be intensified regarding the lay out of TFT circuits.

REFERENCES

(1) S.Morozumi, K. Oguchi, T. Misawa, R. Araki, H. Okshima, SID 84 Digest, 316 (1984).
(2) e.g. R. Paul: "Mikroelektronik", A. Hüthig Verlag Heidelberg (1981).
(3) C. Hilsum et. al., ESPRIT '86: Results and achievements. Elsevier Science Publishes B.V. (North Holland) 1986.
(4) Y. Oana, H. Kotake, N. Mukai and K. Ide, Jpn.J.Appl.Phys., 22 Suppl. 22-1, 493 (1983).
(5) M. Matsui, Y. Shiraki and E. Maruyama, J.Appl.Phys. 55, 1590 (1984).
(6) B. Loisel, P. Joubert, L. Haji and Y. Chouan, Proceedings of the International Symposium on Trends and New Applications in Thin Films (Societe Francaise du Vide, Strasbourg, 1987) pp. 249-253.
(7) D. Meakin, J. Stoemenos, P. Migliorato and N.A. Economou (to be published in J.Appl.Phys.).
(8) G. Harbeke, L. Krausbauer, E.F. Steigmer, A.E. Widmer, H.F. Kappert and E. Neugebauer, J. Electrochem.Soc. 131, 675 (1984).
(9) R. Bisaro, J. Magarino, N. Proust and K.Zellama, J.Appl.Phys. 59, 1167 (1986).
(10) B. Loisel, Y. Chouan, N. Pedrono, Electr.Letters 23, 288 (1987).
(11) S.M.Sze: "Physics of Semiconductor Devices, Wiley-Intercience New York (1969).
(12) J. Vanfleteren, A. Van Calster and H.J. Pauwels, Intern.Display Research Conf. (San Diego, 1985), 72.
(13) T. Noguchi et al. Jpn.J.Appl.Phys. 25, 2121 (1986).
(14) T. Oshima et.al.Jpn.J.Appl.Phys. 25, 2291 (1986).
(15) N. Shibata, K. Fukuda, H. Ohtoshi, J.Hanna, S.Oda and T. Shimizu, Jpn.J.Appl.Phys. 26, 210 (1987).

Project No. 1270

ADVANCED PROCESSING TECHNOLOGY FOR GaAs FIELD EFFECT TRANSISTORS AND LASERS

A. Christou, N. Papanicolaou and Z. Hatzopoulos

Research Center of Crete, P.O. Box 1527, 711 10 Iraklion, Crete, Greece

ABSTRACT

We present results of the ESPRIT Program 1270. Advanced processing technology has been developed using molecular beam epitaxial growth of GaAs and GaAlAs in conjuction with laser processing using excimer (UV) laser. These MBE and excimer laser processing techniques are developed for modulation doped transistors and superlattices.

MBE growth parameters have been determined with and without laser interaction. MBE layers have been characterized through mobility measurements, photoluminescence, SEM and TEM. HEMTs have been fabricated and tested at high frequencies and have shown 50% power efficiency.

Studies have also been carried out on refractory silicides and amorphous metallizations. Amorphous films of W-Si and TiW-Si have been achieved on GaAs and InP using thin sputtered layers. Schottky barriers of W-Si and TiW-Si have been developed for GaAs. Metal silicides have been annealed with excimer laser and have been studied with SEM and EDS X-ray analysis.

I. MBE GROWTH

MBE growth optimization for modulation doped field effect transistors has been determined. These conditions using initially a laser desorption of 1 pps, 15 pulses at 248 nm and 90 mJ/cm^2 were analyzed to be consistent with maintaining a flux ratio of As_4/Ga of 5 and a substrate temperature of 610°C. These initial desorption conditions resulted in a very thin 10-20 Å GaO on the surface which was readily desorbed at 580 °C to obtain a reconstructed 2x4 surface. After MBE growth oval defect density was less than 100 cm^{-2}, and dislocation density was maintained less than 10^3 cm^{-2}.

In terms of laser processing optimization, the laser desorption experiments have been carried out at 248 nm, with the laser beam energy density varied from 75-165 mJ/cm^2. The optimum energy density for laser desorption was determined to be between

75-90 mJ/cm^2. Mass spectroscopy analysis indicated that the desorbed species to be GaO. At energy densities above 90 mJ/cm^2, As_4 desorption from the substrate was observed indicating the occurrence of surface damage.

The grown multilayer structures were analyzed by mobility measurements, photoluminescence and scanning electron microscopy. In terms of optimizing the layered structures for high mobility transistors, two spacer regions were used with an intervening region of ten periods of GaAs/AlGaAs. Figure 1 shows the relationship between mole fraction aluminum, electron concentration and Hall mobility. An electron mobility of 131,000 cm^2/V-s was obtained at 77 K with an electron sheet concentration of 1.2×10^{12} cm^{-2}. In order to assess the optical quality of the $Al_{0.41}Ga_{0.59}As$ and $Al_{0.22}Ga_{0.78}As$ layers, photoluminescence spectra of $Al_xGa_{1-x}As$ layers were taken at 4.2 K as a function of mole fraction of aluminum. The photoluminescence spectra exhibited a bound exciton peak (D-A pair) at 1.846 eV. At x=0.41, a half-width at full maximum of approximately 5 meV was obtained, while at a mole fraction of 22%, the half-width at full maximum of the DX exciton was measured to be 3-3.5 meV, which remained constant up to a mole fraction of 33%. The photoluminescence (PL) spectra for AlGaAs were indicative of the low defect concentration material necessary for providing the potential barrier for electron confinement in order to obtain a high electron mobility at 77 K.

The growth of GaAs on silicon has been carried out by MBE and MOCVD. The MBE technique involved an initial laser induced desorption at 105 mJ/cm^2 and 135 mJ/cm^2 at a wavelength of 248 nm. The grown GaAs surfaces were very smooth with an oval defect density of approximately 2000 cm^{-2}. Field effect transistors processed on these layers also included an GaAlAs undoped buffer layer between the first 2000 Å GaAs layer and the top active GaAs layer. The transistors are shown in Figure 2 for various GaAlAs thickness and indicates that the gate threshold voltage to be between 0 volts and -1.5 volts. The transistor characteristics are well behaved with no indication of a backgating phenomenon.

The GaAs growth process consisted of initially depositing an amorphous GaAs layer at 250°C and then increasing the substrate temperature to 650 °C. Thus the initial amorphous layer was recrystallized during the final deposition state. The growth of GaAs was successfully accomplished without the presence of anti-phase domains. The laser annealing they may have produced a two step surface which prevented antiphase domain growth.

II. PROCESSING AND ANALYSIS

MBE GaAs field effect transistors and high electron transistors were processed on layers provided by the Research Center of Crete. In addition FETs were also processed on MBE GaAs/Si and laser recrystallized GaAs on silicon layers. The results reported by CNET for RCC MODFETs shows a 1.4 dB noise figure at 10 GHz and 11.5 dB associated gain. The multi-channel MODFETs show a 50% efficiency at 12 GHz and 26.5 dBm of output power. These results are better than presently available commercial transistors.

Field effect transistors were also processed on recrystallized (100)GaAs on (100)silicon. Transistor IV characteristics shows a transconductance of 40-45 mS/mm and does show that transistor-like behavior is possible on recrystallized GaAs layers with low background doping concentration. These transistors fabricated are in addition to the MBE GaAs/Si layers reported as part section I.

The failure physics investigations are presently underway at CNET and Plessey. Test jigs are in place at both locations and reliability investigations will start during the next quarter.

II.1. Optimised FETs and DFB lasers with Laser Processed Metallizations

Laser processing of WSi_2 and $TaSi_2$ metallizations on GaAs has been carried out. The results show that the inclusion of the silicon structure (WSi_X and $TaSi_X$) dramatically improves the stability of both the elemental W and Ta metallizations. In the case of tantalum silicide, the Rutherford backscattering evidence shows that the interface remains in fact up to 950 °C but shows slight evidence of outward diffusion of substrate components at 1050. The stability of the amorphous silicides have also been found to be highly dependent on the substrate dislocation density. It is shown that low dislocation density GaAs is stable up to 1050 °C. These results indicate the potential for very high reliability Schottky contacts to future GaAs devices.

II.2. Reliability investigations

Test jigs for DC and microwave measurements of MODFETs for up to 40 devices have been fabricated and tested. The aging set up for RF life tests is still under fabrication and is based on a 12 port power divider-combiner. The test-jigs for laser characterization have been fabricated and tested. The aging set-up for lasers is under final assembly.

II.3. Heterojunction Models

A heterojunction model for GaAs/GaAlAs and GaAs/Si structures has been set-up in the RCC VAX computer. It is based on a static model taking into account measured band-gap discontinuities and built-in potential differences between GaAs and GaAlAs or between GaAs and silicon. From this model GaAs/Si diode IV characteristics have been estimated. The development of the necessary transport equations will be accomplished in the subsequent phase.

III. CONCLUSIONS

The effort for the first eight months of the ESPRIT project 1270 has focused on molecular beam epitaxial growth, the development of laser processing techniques and laser processing of amorphous metallizations. Also developed were single and multiple channel MODFETs which exhibited significant improvements in power added efficiency in comparison with state of the art MODFETs.

Significant progress is also reported on GaAs/Si FETs with laser processed GaAs on silicon FETs initially processed with an excimer laser at 248 nm. Available for further investigation of device structures are both MBE and MOCVD structures of GaAs on silicon.

ACKNOWLEDGEMENTS

The results of this project is due to the enthusiastic work of all of the project partners: RCC, Plessey, CNET (Lannion), UWIST and ELLTEC.

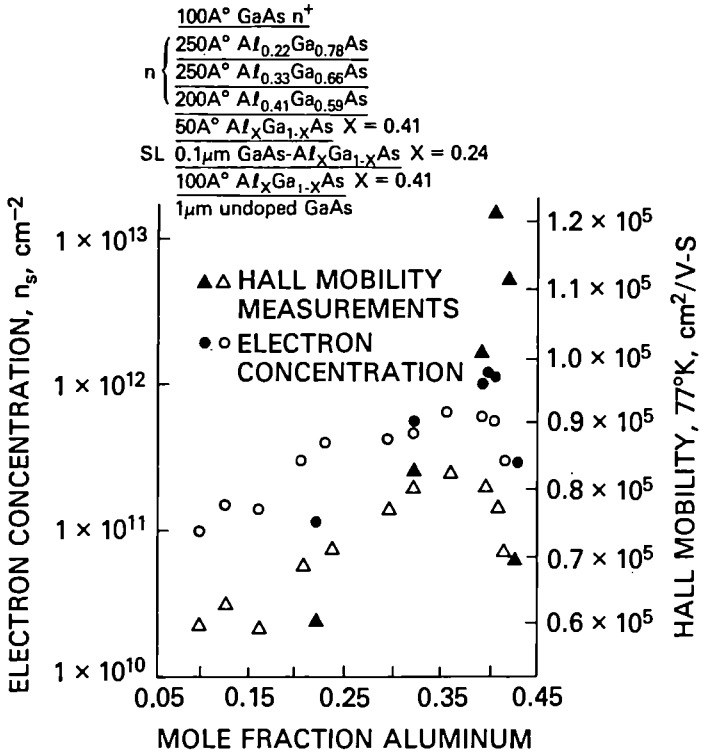


FIGURE 1 The relationship between mobility and electron concentration for various Al concentrations.

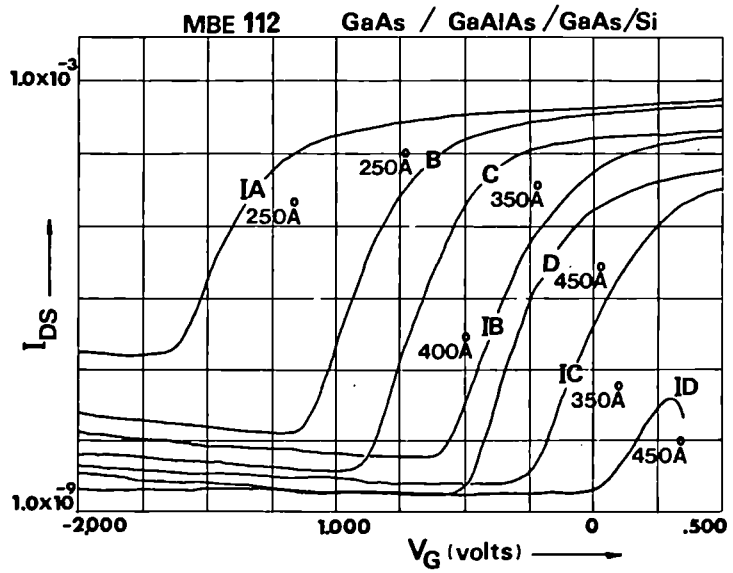


FIGURE 2 Current-voltage characteristics for GaAs on silicon.

Project No. 334

PLASMA DEPOSITION TECHNOLOGY FOR MAGNETIC RECORDING THIN FILM MEDIA

B. Cord, P. Wirz
Leybold-Heraeus GmbH, Wilhelm-Rohn-Str. 25, 6450 Hanau, FRG

1. OBJECT OF THE WORK

A significant increase in magnetic recording density can be achieved if the particulate magnetic coatings are replaced by continuous magnetic thin film media.

In numerous experiments thin film media have been prepared and tested. Most often the thin films were produced in laboratory scale or with low deposition rates for example by rf-diode sputtering with 0.5 nm/s. The present project aimed to the development of vacuum deposition technologies with high deposition rates for industrial production of thin film media.

- Magnetron sputtering with a rate of 10 nm/s.
- Electron beam evaporation with rates up to 100 nm/s.
- Development of inline deposition systems with a high throughput for large scale production.
- Development of layer systems on floppy and rigid disks with the appropriate qualities for high density recording.

In the ESPRIT project three industrial partners work together to establish this technology for the European market. Leybold-Heraeus (FRG) as the prime contractor is in charge of the development of various deposition technologies - sputtering, evaporation and plasma-CVD. BASF (FRG) is responsible for aspects of media fabrication and evaluation of differently prepared media. BULL (France) has large experience with respect to media available on the market and the implementation of them into a disk drive system.

The work is supported by academic institutions in regard to scientific investigations. In addition the University of Twente, Enschede (J.C. Lodder), (The Netherlands) developes an evaporation technology in laboratory scale for CoCr-vertical recording.

2. PREPARATION AND EVALUATION OF RIGID DISKS FOR LONGITUDINAL RECORDING

(Leybold-Heraeus, BASF, Bull)

Rigid disks with thin magnetic films are already in the market. The samples are prepared by either plating or sputtering or a combination of both. Often the magnetic layer is plated and the protective layer is sputtered from a carbon target. Within the current project a process technology for full sputtered disks is developed, since it is expected that the full sputtered media are coming more and more into the market.

The layer stack consists of a Cr-underlayer, a CoNi(Cr) magnetic layer and a protective carbon overcoat. An in-line sputtering system for rigid disks, capable for the production of 200 disks/hour was put into operation and the process technology was developed.

Fig. 1 gives a schematic drawing of the used system. The substrates are set onto a palette and moved through separately working process chambers without breaking vacuum.
The different chambers allow pretreatment (e.g. cleaning, heating) or sputtering of material. The complete layer stack is brought onto the disks in one run.

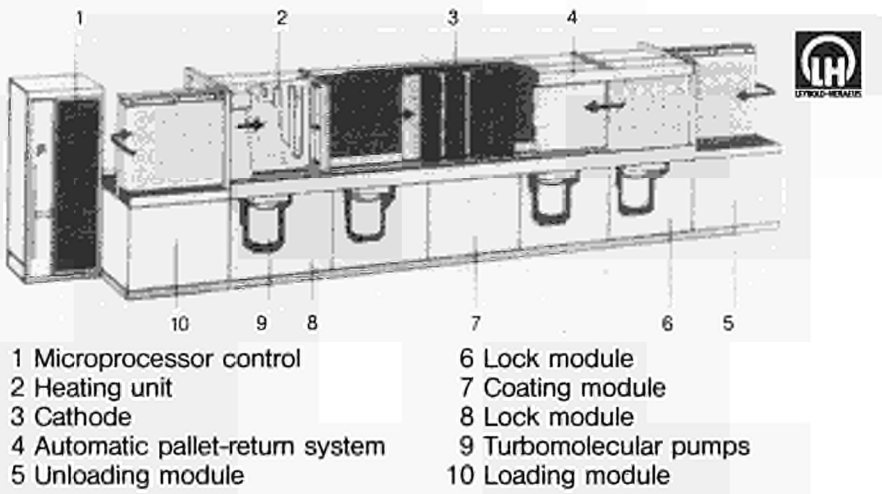


FIGURE 1
In-line sputtering system for the production of rigid disks

As an example for the developed process technology fig. 2 shows the dependence of coercivity of the magnetic CoNi(Cr)-layer on the thickness of a previously sputtered Cr-underlayer. Cr-, CoNi(Cr)-targets with a size of 125 x 750 mm² were used with deposition rates of 8 nm/s.

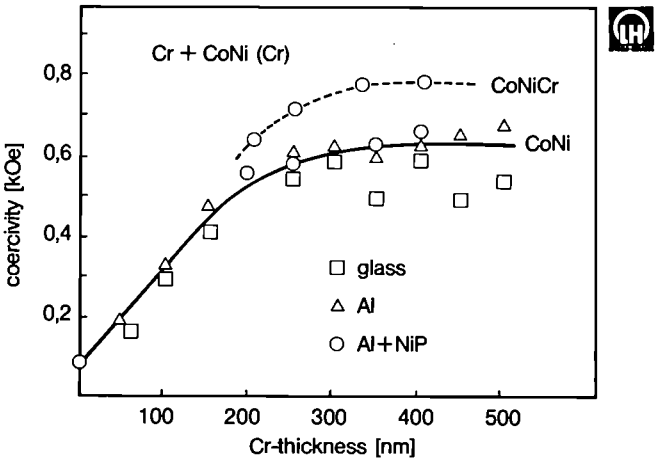


FIGURE 2
Dependence of the coercivity of the magnetic CoNi(Cr) alloy on the thickness of a Cr-underlayer. Several substrates were used. The magnetic layer thickness is kept constant and amounts to 70 nm for CoNi and 90 nm for CoNiCr.

Samples prepared within this machine were given to the partners (BASF, Bull) for testing the electromagnetic data and fig. 3 gives the dependence of data on the thickness of the magnetic layer.

An unwanted modulation of the readback signal is obtained in in-line sputtering systems. The linear movement of the substrate relative to the cathodes induces a magnetic anisotropy with an easy magnetic axis parallel or perpendicular to the movement of the substrates leading to a 180° periodic modulation of the readback signal as shown in fig. 4. A special process technology was developed, which allows the production of nearly modulation free disks (fig. 4).

The process technology and the testing of the electromagnetic data has developed so far that the desired magnetic values as coercivity, squareness, signal amplitude etc. can be selected - within the physical interdependencies - by appropriate choice of the process parameters.

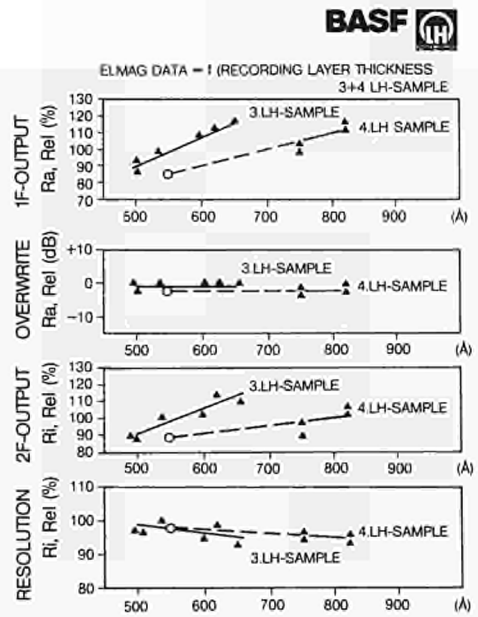


FIGURE 3

Electromagnetic data (1F-output, overwrite, 2F-output, resolution) as function of magnetic layer thickness
3. LH-sample: CoNi (80/20)
4. LH-sample: CoNiCr

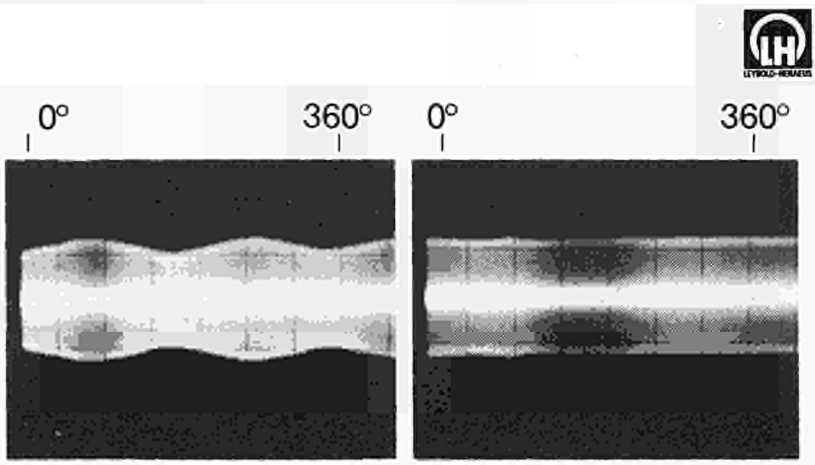


FIGURE 4

High modulation case

No modulation due to special process technology

Modulation of the read-back signal amplitude from sputtered disks produced in in-line system.

Fig. 5 gives a comparative study of heads performed by BULL. Different heads are compared on the same disk in respect to signal amplitude, D_{50} value of recording density and track width.

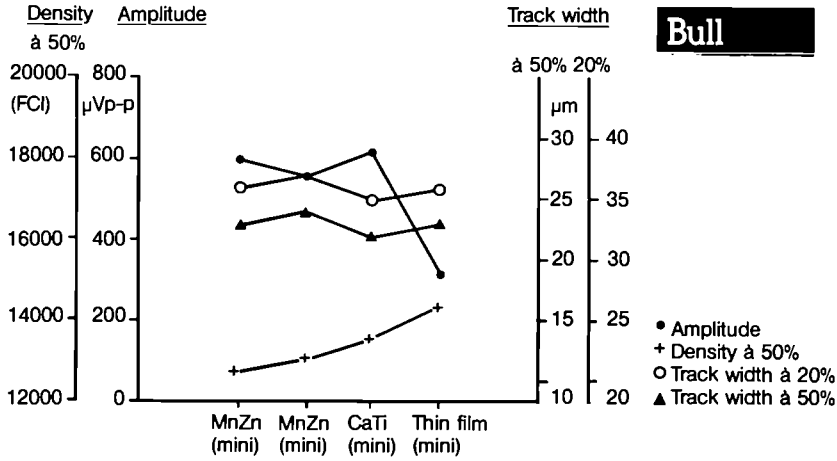


FIGURE 5

Comparison of heads

Looking for the dispersion of one kind of heads the mini-monolithic heads are more homogeneous (amplitude, S/N, resolution) than mini-composite heads. Thin film heads show a low amplitude value but a good resolution of D_{50}-values greater than 23 kfci.
Comparing disks the plated ones exhibit a low modulation and S/N ratio with a good dispersion, the plated + carbon overcoat show good homogeneity of amplitude but dispersion of resolution and S/N ratio. The full sputtered media have generally a higher dispersion of samples from one lot but can achieve good results. For plated, plated and carbon and full sputtered disks there exists not much difference between the average electromagnetic values. Compared to the oxyde media the new thin film media are superior in the electromagnetic data (higher signal amplitude, 10 % higher resolution, better overwrite etc.)

The disks produced in the project are compared to disks available on the market to give an actual picture of the state of art. At the moment a standard of the disks from the project is reached, which corresponds the common standard. A further progress is expected from some modifications of the deposition process, optimized substrates and heads.

3. EVAPORATION OF CoNiCr FOR LONGITUDINAL RECORDING THIN FILMS ON FLEXIBLE DISKS

(BASF)

The performance of a thin film flexible disk for longitudinal recording has been studied. An alternative route to high density longitudinal recording on rigid disks or CoCr-vertical recording on flexible disks was considered [1].

The disks of a CoNiCr magnetic layer on a Cr sublayer were prepared by e-beam evaporation from two independent crucibles with Cr and CoNi (80/20). The Cr-sublayer was prepared by evaporation from the Cr-source and the CoNiCr by parallel evaporation of both crucibles.
Different types of polyimide films with thicknesses of between 30 and 75 µm were used as base material and outgased by additional heat treatment at 150 °C in the vacuum chamber.
At a constant thickness d = 50 nm of the CoNiCr-layer the thickness of the Cr-underlayer was varied and the results are given in fig. 6. With increasing Cr-thickness the magnetic data improve, but the flexibility of the complete disk is reduced leading to problems in maintaining good head-disk contact. The thinnest Cr-layer (200 nm) compatible with the magnetic requirements was chosen. The dependence of H_c and remanent flux on the CoNiCr-thickness is given in fig. 6, too.

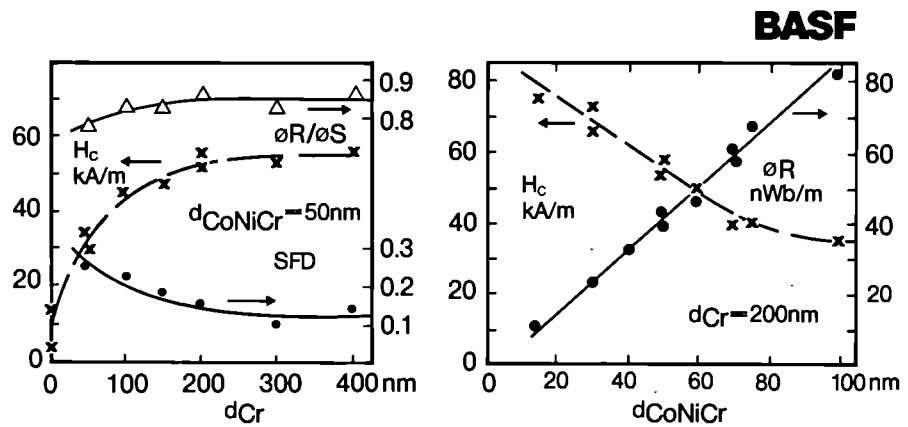


FIGURE 6

Coercive force, squareness and SFD as functions of the Cr-sublayer thickness. The magnetic layer thickness is kept constant.

Coercive force and remanent flux for different thick CoNiCr-layers on a 200 nm thick Cr-sublayer.

The flexible disks were coated with a sputtered protective carbon overcoat, mounted in a conventional cartridge and tested in a standard drive system fitted with a special magnetic head. The extremely short head gap of 250 nm was chosen to take full advantage of the high density potential of the disk. With the optimized disk a D_{50} = 62 kfci was achieved.

One of the strongest factors effecting D_{50} is the head to medium spacing, which is demonstrated in fig. 7 by the experimental dependence of D_{50} on the thickness of the carbon top coat. Therefore thin top coats are favourized for high density recording performance.

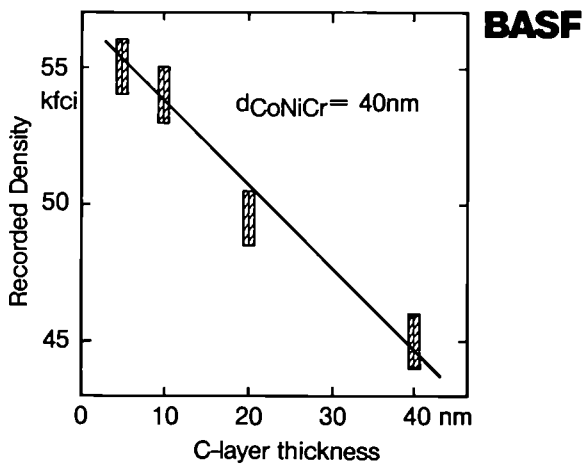
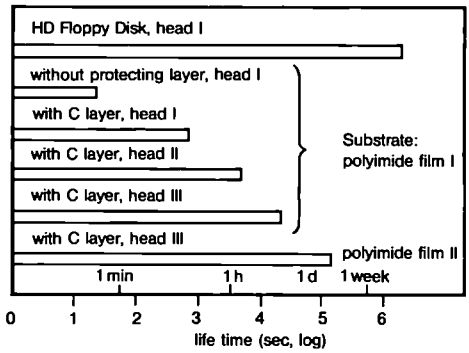


FIGURE 7
D_{50}-values versus the C-layer thickness. The film substrate, the CoNiCr-thickness (40 nm) and Cr-sublayer (200 nm) were constant.

The lifetime of the disks was tested with different head configurations and sputtered carbon as protective overcoat (fig. 8). With a double head assembly (Head I) and an unprotected CoNiCr-disk the lifetime (defined as the time the output level decreases to 90 % of its original value) amounts to about 40 s. A spherical single sided head (Head III) and a sputtered carbon overcoat led to the longest lifetime.



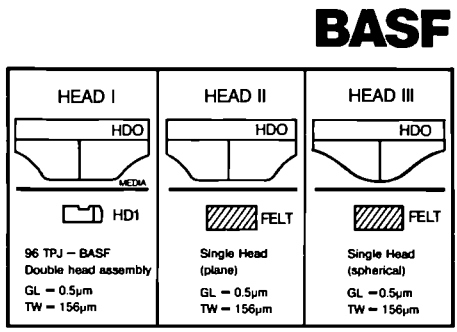


FIGURE 8

Logarithmic lifetime for different CoNiCr-disks compared with a standard particulate medium. C-layer is always 40 nm. The polyimide films differ in the roughness.

Schematic view of the head configurations used in the lifetime test.

The experiments have demonstrated that high density recording on flexible disks prepared by CoNiCr-evaporation is promising. The electromagnetic results are excellent, the lifetime is acceptable but has to be optimized further using other top coats, head configurations and base films.

4. CoCr-SPUTTERING IN LARGE SCALE ROLL COATING SYSTEMS

(Leybold-Heraeus, BASF)

A double side sputter roll coater for 1.2 m web width has been used to sputter CoCr onto polyimide web. The production of sputtered CoCr films in production scale was demonstrated. The magnetic as well as crystallographic properties have been investigated as function of the deposition parameters, layer thickness, coating drum temperature, background pressure and angle of incidence [2, 3, 4].

Fig. 9 shows a sketch of the roll coating system.

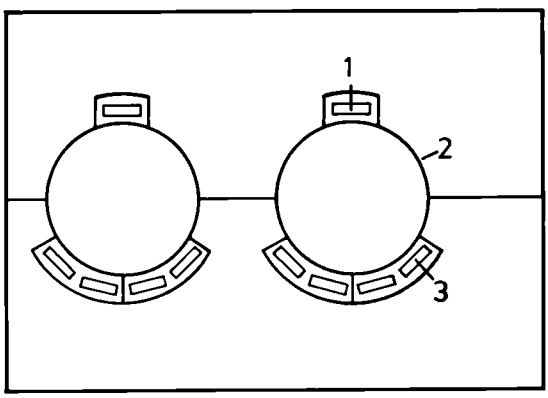


FIGURE 9
Schematic drawing of a 1,2 m two side sputter roll coater for perpendicular recording media manufacturing
(1: glow discharge, 2: cooling drum, 3: cathode)

It can be equipped with 8 DC-magnetrons of 1.5 m length. The two coating drums allow double side coating with a complete layer stack for vertical recording on both sides of the foil. A special magnetron cathode for sputtering thick magnetic material was designed and tested in this machine (fig. 10). The target design overcomes the problems own to sputtering of magnetic material. For ferromagnetic targets the magnetic fields are short-circuited and therefore magnetron sputtering is prevented. The new cathode guides the magnetic field in such a way that magnetron sputtering with rates of 19 nm/s for CoCr is possible. Thick target material can be used with a high target utilization of 53 % compared to 20 - 30 % for the standard version.

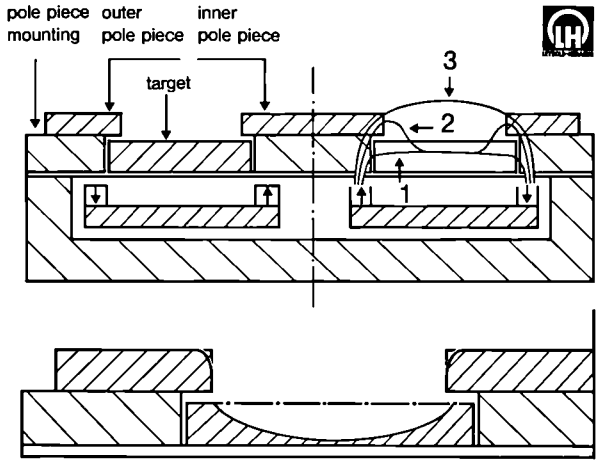


FIGURE 10
Magnetron configuration (top) and erosion profile (bottom) of cathods for sputtering magnetic material
(1, 2, 3: typical magnetic field lines)

CoCr (81/19) was sputtered onto polyimide web, the anisotropy field H_K and the coercive forces $H_{c\perp}$ (perpendicular) and $H_{c//}$ (inplane) were taken from VSM measurements. The main parameter which influences the layer quality is the substrate temperature.

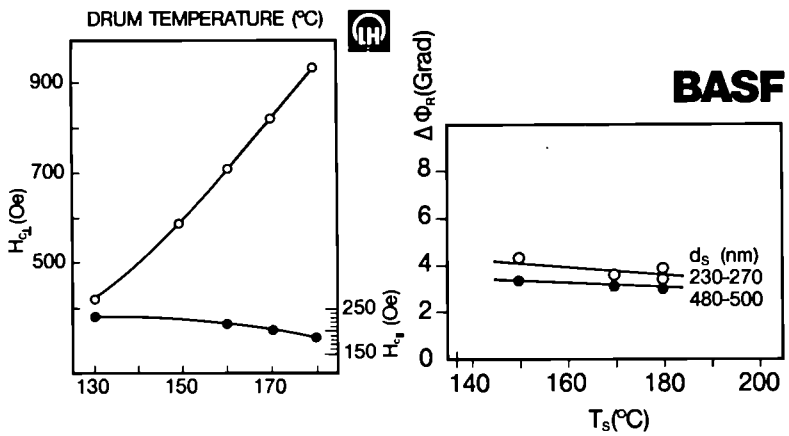


FIGURE 11
Perpendicular ($H_{c\perp}$) and parallel ($H_{c//}$) coercivities of a sputtered 260 nm CoCr-layer as function of substrate temperature

Dependence of the $\Delta\Theta_{50}$-value of the Co (002) peak on substrate temperature during sputtering

Fig. 11 gives the coercive forces as a function of the coating drum temperature for 260 nm thick CoCr layers. A high anisotropic film is obtained at elevated substrate temperatures.

The good crystal orientation could be confirmed by X-ray rocking curves of the (002) Co peak. The right part of fig. 11 gives the $\Delta\Theta_{50}$ values for the left samples. Values as low as 2.6° were obtained.

For recording experiments 5,25" disks were punched out from the polyimide web with the following magnetic layer parameters:

$$H_S = 3550 \text{ Oe}, \quad H_{c\perp} = 600 \text{ Oe}, \quad \Delta\Theta_{50} = 3.4°$$

A special video ring head with a gap length of 0.25 µm was used.

The frequency response for the disk with a liquid lubricant of 2 nm is shown in fig. 12. For this small head medium spacing D_{50} as high as 90 kfci could be measured. In increase of the spacing for example by a carbon coating reduces the D_{50} value considerably.

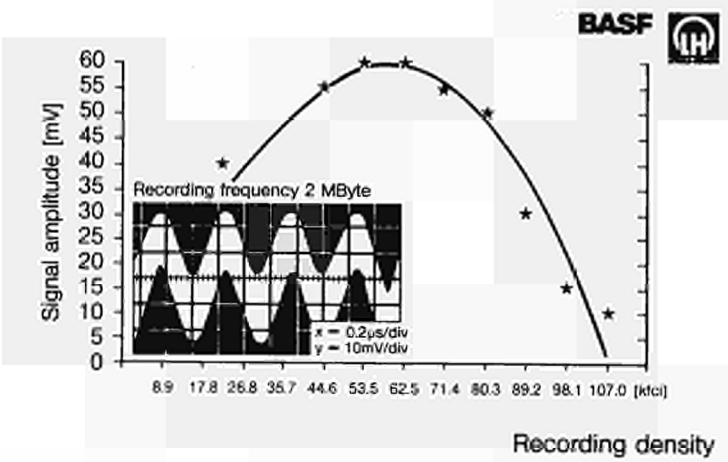


FIGURE 12
Frequency response of a CoCr flexible disk. The insert shows the signal at 2 MHz

The current work has demonstrated that CoCr-layers can be sputtered in a large scale production system. The achieved magnetic and crystalline properties are suitable for production of perpendicular recording media.

5. EVAPORATION OF CoCr

(University of Twente, Enschede, The Netherlands)

In this task Co-evaporation experiments for preparation of CoCr-layers for vertical recording are performed. The main goals are:

- The perpendicular magnetic anisotropy

- The set-up of CoCr evaporation technology.

The deposition is done in a Leybold-Heraeus L 560 vacuum system, equipped with two e-beam evaporation sources. The angle of incidence on the substrate is 20 - 30° but from opposing directions for Co and Cr.
The experiments are projected as an alternative to sputtering of CoCr.
The work is performed at laboratory scale and the fundamental effects such as deposition rates, areal composition homogeneity, angle of incidence effect, influence of substrate temperature, growing characteristics are under investigation.
The experiments started in January 1987. CoCr-layers with the desired perpendicular orientation could be prepared [5]. At base pressures $< 2 \times 10^{-7}$ mbar with deposition rates of 0.8 nm CoCr-layers of about 200 nm were deposited by simultaneous evaporation of Cr and Co.

Fig. 13 shows the dependence of the saturation magnetization M_s as function of Cr-content. At higher process temperatures T_p M_s increases which is explained by a pronounced inhomogeneity of the local Cr-concentration [6].

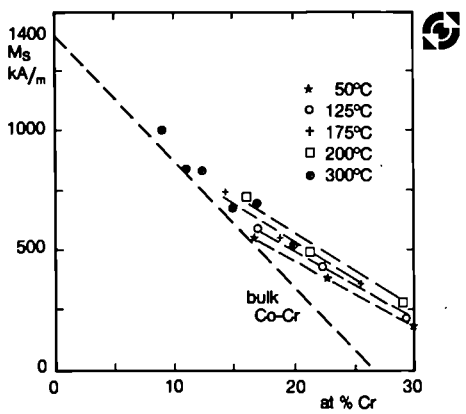


FIGURE 13
M_s as function of Cr-content
Straight line: Value for bulk material
Data points: From evaporation experiments at different substrate temperatures

In addition the evaporated CoCr-layers show a considerably increased M_s as compared to bulk CoCr. This indicates that Cr-inhomogeneity may not only be reached by elevated substrate temperatures. An additional process induced Cr segregation happens at intermediate oblique incidence of the incoming Cr- and Co-atoms. The inhomogeneity in the layer is expected to be due to shadowing during the growth process. The further experiments should give more detailed pictures of CoCr-growing.

It is planned to modify the experimental set-up in this L 560 laboratory system in order to allow the fabrication of a flexible disk. This CoCr-disk could then be tested for read/write characteristics.

6. SUMMARY

Starting from laboratory results industrial vacuum deposition technologies both for longitudinal and vertical thin film recording media were developed within the ESPRIT-project.

The magnetron sputtering has reached the highest standard. Pilot production machines were built up and samples were produced.

The e-beam evaporation is run in laboratory scale.

- Large scale magnetrons up to a size of 150 x 28 cm^2 with deposition rates up to 19 nm/s have been constructed and were tested in production plants.

- A production like in-line sputtering system for hard disks was built up. The machine coats 200 5 1/4"-disks/hour with a complete layer stack - underlayer, magnetic layer, overcoat. By the selection of optimized target material and the good control of layer composition the sputtering machine is very flexible and has therefore a clear advantage compared to thin film plating technology.

- Rigid disks for longitudinal recording from the pilot production system were tested and compared to disks available on the market. A standard is reached which corresponds to other thin film media and is therefore higher than for oxyde particulate media.

- Flexible disks for longitudinal recording were coated by evaporation. Using special heads high recording densities and acceptable lifetimes were achieved demonstrating the great potential of this medium.

- The practical limits of high density longitudinal recording were tested from the viewpoint of a diskdrive manufacturer.

- The deposition technology of CoCr for vertical recording was developed in a large scale sputtering roll coater for industrial production of CoCr-layers on foil. Read/write procedures were run on flexible disk samples.

- The preparation of vertical CoCr-layers by e-beam evaporation was performed in laboratory scale and the results are compared to sputtered CoCr.

The future activities will concentrate on further improvements of the sputtering technology and the comparison of the deposition techniques. In addition the process technology of protective layers by plasmapolymerization will be developed. Advantages compared to sputtered carbon overcoats are expected.

REFERENCES

[1] CoNiCr Thin Film Flexible Disk for Longitudinal Recording
M. Hitzfeld, B.K. Dalmann, H. Jakusch
Paper at the Intermag Conference 1987, Tokyo

[2] High Rate Sputtering of CoCr with Large Magnetrons - Dependence of Magnetic Properties on Sputtering Parameters
R. Ludwig, K. Kastner, R. Kukla, M. Mayr
IEEE Tran. Mag., Vol. MAG-23, no. 1, 94 (1987)

[3] High-Vacuum-Sputter Roll Coating for Production of Magnetic Recording Media with Perpendicular Magnetisation
M. Mayr, K. Kastner, R. Kukla, R. Ludwig
IEEE Tran. Mag., Vol. MAG-23, no. 1, 131 (1987)

[4] CoCr-Sputtering in Large Scale Roll Coating Systems
M. Mayr, R. Ludwig, K. Kastner, R. Kukla, B. Dalmann, H. Haberkorn, M. Hitzfeld
Paper at the Intermag Conference 1987, Tokyo

[5] Co-evaporation of Co-Cr at intermediate oblique incidence
F.A. Pronk, J.C. Lodder
Paper at the EMMA 87 Conference, Salford, Sept. 1987

[6] Y. Fujii, K. Tsutsumi, T. Mumata, Y. Sakurai
J. Appl. Phys. 55 (6), 15 March 1984, p. 2266

Project No. 443

NEW HORIZONS FOR THE CHEMICAL INDUSTRY IN INFORMATION TECHNOLOGY

J. ZYSS (*)
CNET (*) prime contractor (France), ICI (U.K.), Thomson-CSF (France), FUNDP (Belgium)

ABSTRACT
Optics and microelectronics will increasingly interact in the form of active or passive optical interconnects between microelectronics chips, mixed opto-electronics devices, and purely optical devices as part of mixed opto-electronics systems. Highly nonlinear optical materials to operate, in proper environment, as optical logic elements, large band modulators, tunable parametric amplifiers or emitters in digital or analogous information processing and transmission systems are therefore required. Optical nonlinearities may originate from quasi-resonnant interaction of light with highly confined states in low-dimensional semiconductor structures such as multiple-quantum-well of III-V compound composition or, alternately from non-resonnant tunable parametric interactions with highly delocalized and polarizable electronic systems. The latter approach has been chosen in this Project where lower non linearities are traded for ultrafast (i.e. quasi instantaneous) response and recovery times, transparency and wide-band tunability. Long conjugated chains can be synthetized at will and their properties finely tuned by molecular engineering such as allowed by the unlimited possibilities of organic synthesis. In the four Tasks assigned to this Project, significant milestones have been reached, based on a truly collaborative effort between fundamental and industry oriented partners. A computer aided crystal engineering workstation has been defined and implemented : its role is central in the optimization of the structural and electronic properties at either microscopic or macroscopic levels. Significant advances have been reached in the synthesis and poling of new molecular materials, such as guest-host liquid crystalline polymers with measured nonlinearities one order of magnitude above that of $LiNbO_3$. Langmuir-Blodgett technology as an alternate for organics to epitaxial deposition has been extensively explored and extremely high susceptibilities (up to 10^{-27} esu) measured in waveguiding structures. It is hoped that future work, within the Phase II framework, will be allowed to build on this know-how basis towards the development of a molecular based Information Technology devices.

I. Purpose and environment : an industrial perspective

The heavy chemical industry specializing in the mass-production of low value added rough commodities is currently meeting a worldwide crisis, especially acute in Europe. This traditionnally important sector of activity is irreversibly shifting both structurally and geographically, to the petroleum and gas industries outside of the EEC sphere. This situation leaves no other option in the long range to the chemical industry but to seek new products and markets openings where its traditionnal skills may be further fruitfully exploited on a sound economic basis. Besides biotechnology and pharmaceutics which fall outside our scope, the development of new highly value-added materials, well targeted towards specific applications, such as in demand in the field of Information Technologies (IT) is a highly competitive challenge. In view of its initial assets this challenge may be favourably met by the european chemical industry. On the other hand, the

electronics industry has to prepare for the future, where ever increasing flows of information processed at ever increasing rates will require devices based on materials other than currently used inorganics semiconductors. It is the aim of this Project to combine these two demands into a common exploration by chemists and electronicians of the potential of new tailor-made organic materials for information processing. It is by now a commonly accepted view [1] that optics and electronics will increasingly cooperate in future information processing devices and systems, the trade-off being performed according to the compared relevance of either photons or electrons with respect to specific functions. The barrier is by no means clear-cut and is bound to remain highly sensitive to unpredictable technological advances : for example one may argue wether a Fourier transform, a typical example of a widely used linear processing operation, is more efficiently achieved by means of a computer implemented or wired FFT algorithm or at the focal plane of a converging optical lense. There of course system architectures considerations and cost-effectiveness, in addition to the very nature of the materials and devices involved, will play a pivotal role. In any case, nonlinear optical materials will be part of future optoelectronic information processing or computing devices very much like nonlinearities, resulting from doping and junctions in semiconductors underly present and future purely electronic applications. Such typical nonlinear applications as amplification, logical gating, thresholding modulation, etc... are presently more readily achieved in electronic systems; however, when sufficiently efficient optically nonlinear materials are developped, the well-known additional benefits of optics, as compared to electronics, such as speed, parallelism or bandwidth will drastically modify the scene.

Nonlinear organic materials, based on preliminary investigations as reviewed in Ref. [2-5], prove both a sound and highly promising meeting point answering the previously mentionned driving forces pulling together the chemical and electronic industries. Figure 1 illustrates the intermediary position of an organic material based device industry to be backed at one end by the chemical industries and by the electronics and I.T. industries at the other end.

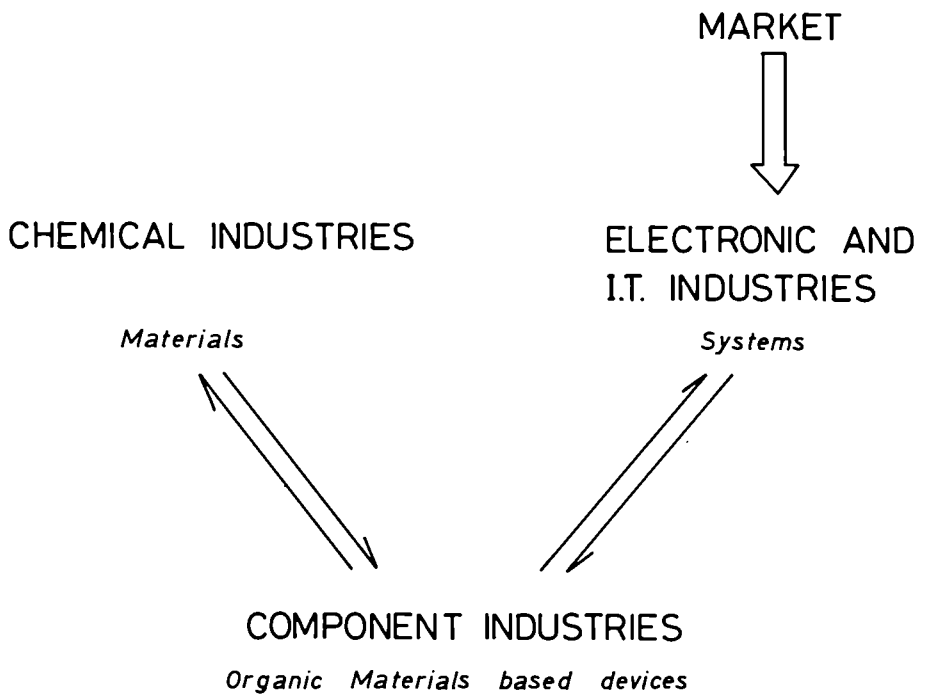


Fig. 1 Industrial environment or Project 443

The expertise, synthetic capability and innovation potential of the chemical industry is highly needed as large numbers of materials will have to be conceived, synthetized and shaped to fulfil the requirements of optimized nonlinear devices following a very similar approach to that used in pharmaceutics. However the approach followed here is based on more precise physical groundrules deserving well the term of "molecular engineering" [3]. The market requirements will be transmitted via the electronic and I.T. industries which are needed for their know-how in components and systems so as to help bridge the gap between materials and systems. The international environment is fast evolving-both in and out of the Community and the effort gathered in this Project ought to be pursued and reinforced to keep-up with the competition and help industrially exploit the predominant initial asset of the european scientific and technical communities as represented within the Consortium. Three kinds of commitments, not necesssarily independant from each other, may be distinguished.

Firstly purely proprietary industrial programmes developped within individual major companies of international stature, tending to keep away for strategic reasons of their own, from external collaborations : Dupont (Central Research Station at Wilmington, Kodak (Kodak Research Laboratories, Rochester) or ATT (Bell Laboratories at Murray-Hill and ATT Engineering Centre at Princeton) exemplify this approach. These companies may occasionnally collaborate with individuals or academic research groups but will not join industrial ventures, their size, own expertise and production branches allowing for self-supported programms integrating research, manufacturing and marketing. Such is also the situation at IBM, within the Almaden Research Centre (San Jose) where a significant but still prospective effort is pursued. Such a strategy remains restricted to truly major companies as can be deduced from the recent withdrawal from that scene of previously active companies of significant size, however of comparatively smaller sizes, such as GTE (Waltham) and Xerox (Rochester). However, limitations resulting from such an approach may be deduced from the recent joint-venture between Dupont and British Telecom which puts Dupont aside in the list. Little can be deduced beyond a rough estimate of the amount of involvement from publications or other types of disseminations as these companies, in consistency with their approach, will not let out much of their achievements. The present tendency is to set-up cooperative organization schemes as will be further alluded to.

Secondly, national programs, gathering at the domestic scale industrial, government-owned and academic laboratories are being set-up. Typical of such an approach are the "Frontier Project" in Japan [6], extending from 1986 to 2001, where one of the seven themes is devoted to "organic nonlinear optics and HTC superconductors", the British Joint Optoelectronic Research Scheme (JOERS) where programms on organic nonlinear materials and Langmuir-Blodgett (L-B) technology are being assembled, the US Optical Circuit Cooperative (OCC) based at the University of Arizona where Celanese is the main chemical industrial participant and which is substantially funded by government agencies. A major development has been the acquisition in 1986 of Celanese by Hoechst which represent an important concentration of skills in organic nonlinear optics. In west Germany, a government sponsored project is gathering a number of major university and Max-Planck research teams around the three major chemical companies Hoechst AG, BASF and Bayer on the theme of functionalized L-B films.

Thirdly, international cooperation, of which little is being known, is connecting american with japanese companies or Dupont with British Telecom. This ESPRIT Project falls within this category in as much as an EEC funded programm may be termed international.

The amount of books, feature issues of international scientific or technical journals, major international meetings concentrating on the subject is increasing fast with a significant presence of participants from this ESPRIT Project. Foremost events in 1987 are the Symposium on electroactive polymers held within the framework of the American Chemical Society national meeting at Denver (april 1987) [7], the Symposium on nonlinear optical properties of polymers held within the Material Research Society meeting at Boston in December [8]. A number of more specialized meetings on L-B films, solid-state chemistry and nonlinear optics are currently being held. No significant advances over the results obtained within our partnership as will be further described, are noted and there is an overall agreement at the present supposedly precompetitive stage, on the avenues to be

explored i.e. : thin films, single crystals, poled liquid crystalline polymers, Langmuir Blodgett technology, nonlinear fibers, as well as on the need for sophisticated and adapted design, growth and characterization tools. It may be noted that a lot of theoretical synthetic and physical expertise built around the theme of conducting polymers, a domain which, although very promising, seems to have reached a plateau, is highly relevant towards cubic optical nonlinearities and is being succefully applied in this newer domain. The university of Bologna, the Santa-Barbara Polymer Institute and our belgian partner (FUNDP) exemplify, among other groups, this exploitation in NLO of these background know-how. The future of this domain now strongly depends on the definition and availibility of organic materials adapted technologies which is a prerequisite towards the development of a fruitful scientific domain on to an industrial market, this being the major issue years to come.

In section II, we will detail the major technical and scientific milestones reached so far in our Project, while section III will be devoted to perspectives.

II. Main achievements over the five first semesters in Phase I (january 1985-june 1987)

These can be classified in three categories namely molecular design and characterization tools, molecules and materials. The breakdown of tasks and major milestones in the Project can be found in Ref.[9].

II.A Molecular design and characterization tools

A condition for further progress in this field, is the availability of various sophisticated tools either existent but requiring higly specialized manpower and lacking the reliability demand in an industrial project (typical of that situation is the Electric-Field Induced Second Harmonic EFISH experiment) or non-existent at the onset of the Project (Electro-Optic assessment of monolayer assemblies). These tools will include molecular material computer aided design systems, on to laser wave mixing facilities and L-B through. Their purpose is to provide the necessary predictive conceptual approaches or experimental feed-back to avoid costly and tedious synthesis dead-ends. Furthermore, the development of such tools into industrial prototypes to participate into future production or testing units is a goal per se. Their availability is viewed in this Project as a "deliverable", endowing the Consortium with the relevent technological environment for further industrial developments.

One of our goals is the devopment, essentially at ICI, of a molecular material computer aided design workstation [10] as permitted by recent advances in the field of molecular graphics over the past decade. These have led to the development of tailored molecular modelling packages which are used on a routine basis in industry to design chemicals for applications in many areas of chemistry, physics, and biology. Typical hardware packages consist of high resolution calligraphic or raster displays such as the Evans and Sutherland PS300 series or the IBM 5000 series coupled to an appropriate host or microcomputer such as a Digital Equipment Corporation Microvax. In the design and engineering of chemicals by computer for applications in electronics the essential software requirements for an effective modelling display (see Figure 2) are the ability to interactively generate a molecule, modify its structure at will, calculate whichever property is of interest, display molecular skeleton or space filling models viewed from various angles, generate the crystal structure from crystallographic data.

For example, the evaluation of the hyperpolarizability of over a thousand different molecules has been achieved by molecular modelling in a minute fraction of the time required for their synthesis and characterization. Candidate molecules have been selected on this basis, synthesized and tested. However, while molecules designed in this way possess large intrinsic hyperpolarizabilities, their behavior in the solid state may be quite different because many active molecules crystallize in a

centrosymmetric fashion and the prized molecular effect is much reduced or even lost. A crystal modelling program has been partially implemented in the course of this project, a major bottleneck being successfully assessed : that is the ability to calculate and predict the crystal structure from hypothetical molecules, a crucial prerequisite to the effective design and engineering of organic devices. Prior to the onset of this Project, it was generally deemed irrealistic, in view of the complexity of the problem, to precisely predict crystalline structures beyond simple trends such as originating from the "closest packing" approach by Kitaigorodsky [11].

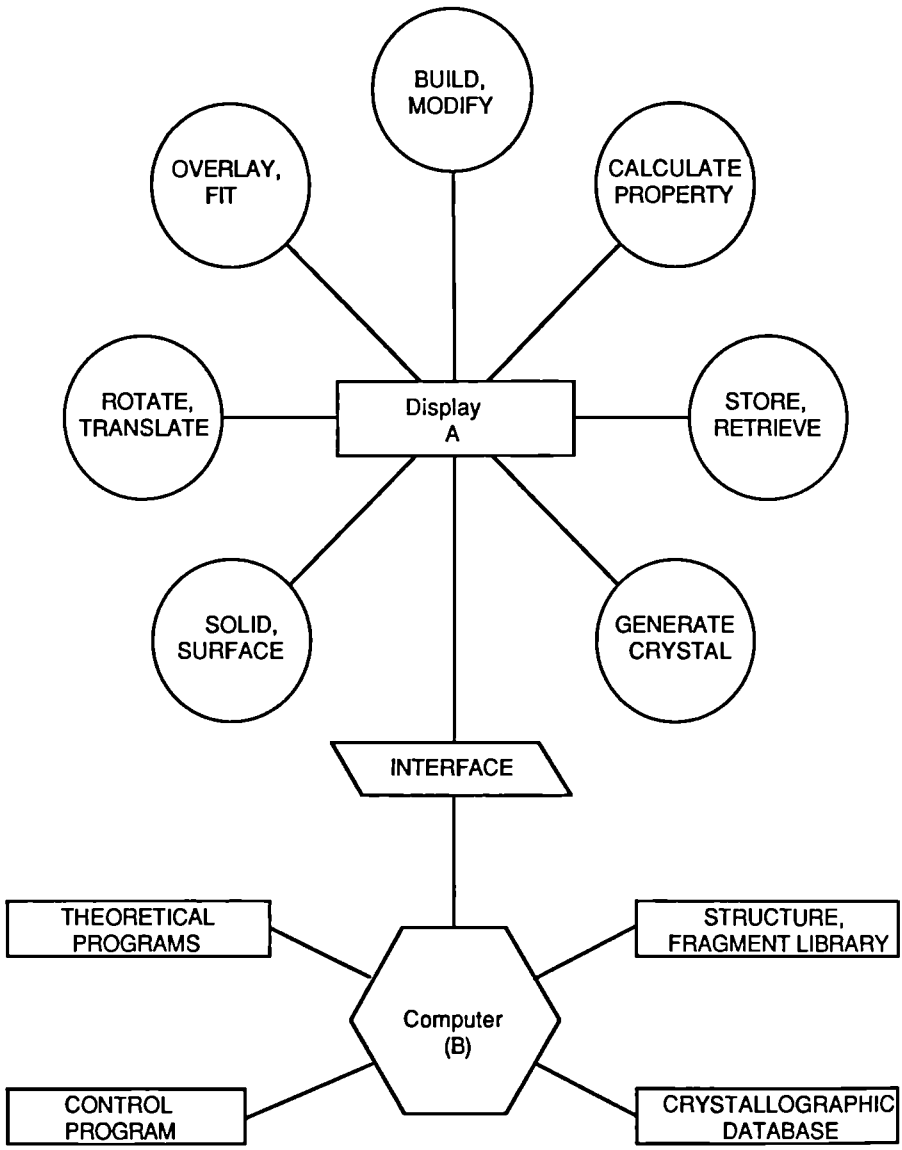


Fig. 2 Molecular engineering workstation for the computer aided conception of new optically nonlinear molecules and materials as from Ref. [10].

Excellent agreement has recently been obtained between the experimental and calculated crystal structures of n,n-dimethyl-p-nitroaniline, with minimal differences in the eleven variables of the

calculated structure and the corresponding experimentally observed values (see figure 3). This agreement supports the use of the calculation procedure in deriving hypothetical crystal structures of n,n-dimethyl-p-nitroaniline by the imposition of alternative symmetries on the arrangement of the molecules in the crystal. The postulated structures are as expected less stable but one is sufficiently close that it may be experimentally accessible.

Preliminary structural calculations on n,n-dimethyl-p-nitro-ethenamine based on similar principles confirm the relevance of this approach and extension to molecules of greater complexity is being undertaken. Previous results on the influence of the nature and length of conjugated systems have been reported [12] and point-out the relevance of polyenes as compared to polyphenyls. The nitroso NO group was also found to have greater electro-attracting potential than the more commonly used nitro NO_2 group. The modelization routines for the computation of β rely on a sum-over-states (SOS) perturbational expression where the molecular quantum eigenstates and eigenvalues diagonalize a semi-empirical Hartree-Fock hamiltonian [13].

The effect of atoms other than carbon on the conjugation path in terms of electron transfer has been investigated for stilbenes, benzylidenes and azobenzenes where the calculations indicate that the azo group is the most efficient conjugated linkage in terms of charge transfer. Interesting structural effects were observed in 1,3-diphenylpyrazolines where the calculated hyperpolarizability is highly dependent on the position of ring substitutents.

Finally as will be seen in Section II-B, the parametrization of the CNDO-VSB method for sulphur has proved consistent with nonlinear measurements at CNET of ICI supplied species.

-The modelization and molecular engineering of molecules, oligomers or polymers with enhanced cubic (γ) nonlinearities, essentially assigned to the FUNDP group consists of two main areas of investigations.

• Firstly, methodological developments, algorithmic design and implementations which were much less advanced for γ than for β at the onset of this Project.

• Secondly, the test of programs and application for searching new compounds, ultimately polymers, characterized by high optical responses and identification of the most important parameters that can be modified to further enhance these responses.

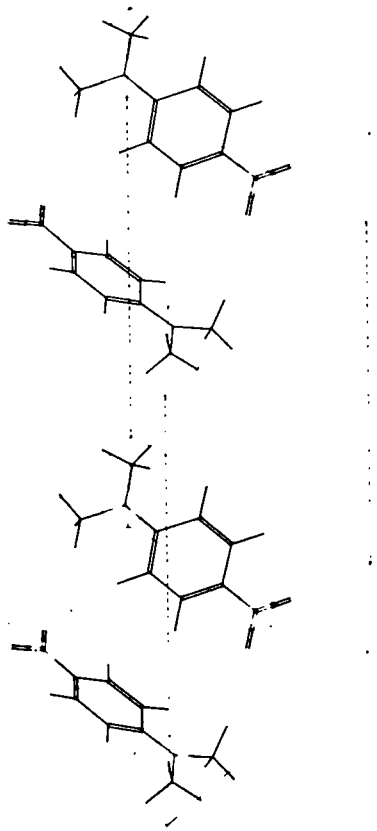


Fig. 3. A comparison of the experimentally observed form of n,n-dimethyl-p-nitroaniline (top view) and the calculated crystalline structure with the same symmetry imposed (Neil Higgins and J.O. Morley, ICI, private communication)

Within the framework of the Hartree-Fock theory,two choices have been considered to calculate the electric polarizability and hyperpolarizabilites of molecules, oligomers and polymers : a)

the sum-over (one-electron) states (SOS) and b) the Finite-Field approach (FF). For each of these choices, specific numerical and programming problems had to be solved [10,14].

More recently, the Genkis and Mednis pertubative approach has been fully mastered [15] and developped from a purely theoretical model on to a practical numerical testing method. It may be viewed as the counterpart for infinite chains of the SOS methods for finite-length systems.

Using the FF method at the ab initio level, the identification of the most promising bonding patterns consistent with a high electric polarizability and the various parameters that can be tuned to further enhance this property has been conducted. From a comparative study on different conjugated subunits, it turns out that the sole number of π-elctron involved is not the only parameter, making obsolete previous "electron in a potential well" type of models. The length is an important parameter to consider since the polarizability increases in a super-linear way with respect to the number of sub-units. From various calculations, the best candidate building bricks for highly polarizable chains are the cumulenic units : -CH=[C=C]=CH- .

The actual geometry of the final system is of great importance. Any effort that can improve on the homogeneity of the π-electron distribution is expected to enhance the resulting electric response. Conversely, any constraint (steric or else) reducing the conjugation between interconnected units has to be avoided. This stresses the importance of being able to control the organization of the molecules and macromolecules in the bulk when considering the synthesis of actual systems. Several case have been successively considered and, FUNDP theoreticians in agreement with ICI organic chemists have agreed upon the concept of "conjugated backbones organized by hydrogen bonds" which is being developped into an organic synthesis program. Further modelization are now in progress on polyaromatic systems, polypyrrole, polythiophene, polyparaphenylene etc,... doped and undoped which can be followed-up in practice by the expertise gathered at CNET (Lannion) initially in view of their conducting or semi-conducting properties.

The setting-up of performant experimental characterization tools as previously mentionned is one of the major objectives of the Phase I of this Project and is being satisfactorily met. A number of specific experiments have been implemented or developped, tested and proved helpful in the task of identifying interesting molecular, thin film or bulk structures, while efficient throughput such as expected in this industry oriented Project will be fully at the end of Phase I. CNET is developping a prototype combined third-harmonic generation (THG) and electric field induced second-harmonic generation (EFISH) set-up for solution testing with tunable fundamental wavelength to allow for the study of systems of various length. Coupling these two experiments is essential, as the sole EFISH yields a combination of γ and β. Disentangling the electronic (γ) and orientational (β) contribution is becoming essential as molecular systems of extended conjugation length, where γ is non-negligible, come into our synthesis programm. Figure 4 shows a simplified view of this experiment which incorporates a SM90, CNET patented [16] mini-computer for driving the equipment and interpreting the data and a Q-switched mode-locked 1.32 μm YAG:Nd^{3+} laser of original CNET conception. This experiment is meant to be transfered to the industry with one technician to man it (instead of two trained chemical-physicists as is the case presently), and a throughput in terms of molecules per day instead of molecules per week as at present.

The electroptic properties of thin layers are being tested by a modified surface plasmon resonnant coupling technique [17,18] (Institut d'Optique and Thomson-CSF). Variations of reflectance, following the modulation by an externally applied voltage on the film deposited on a silver coated prism surface, are being synchronously detected. This allows for the detection of variations on the fifth digit of the index difference between films and substrates.

A pyroelectric measurement set-up has been implemented at Thomson-CSF to help characterize the degree of orientation of polar structures.

Various tools linked to the L-B deposition techniques have been implemented such as dipping troughs at ICI and Thomson-CSF. An original surface-potential measurement has been successfully

set-up at ICI to help define the optimized dipping conditions. It has been proved, in particular, highly sensitive to the crucial phenomenom of bilayer formation at the water-air surface [19].

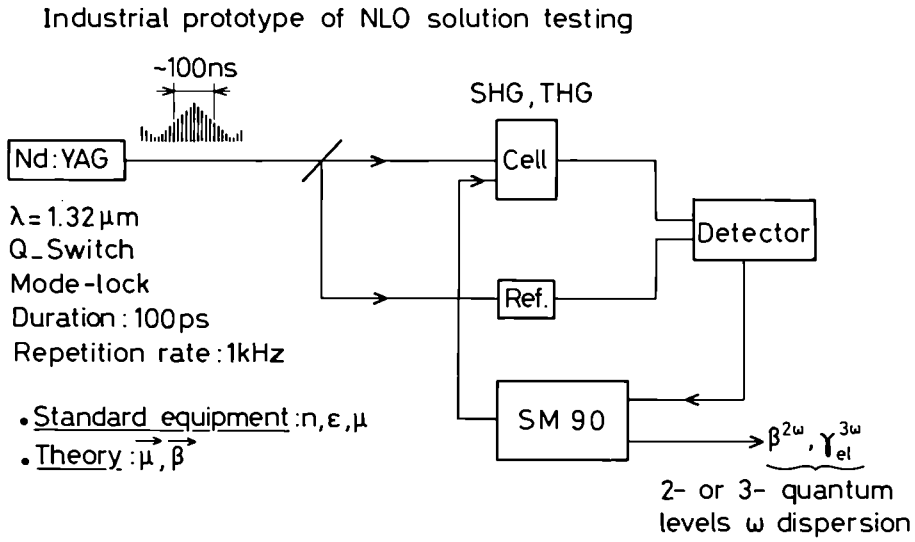


Fig. 4. Project of fully automated prototype of nonlinear assessment in solution of molecular or polymeric species to be transferred to the industry and manned by one technician (CNET)

Testing facilities of the quality and nonlinear efficiency of L-B fims following a method proposed in Ref.[20] and allowing to work close to resonnances has been transferred from CNET to Thomson-CSF. Waveguiding in L-B layers tapped sputtered glass over glass guides as described in Ref.[21], has been undertaken at CNET, using a prism coupling techniques. Adapted complementary ellipsometric measurements were successfully performed in the infra-red and visible ranges.

The femtosecond colliding pulse mode-locked (CPM) laser facility of ENSTA (CNET subcontractor) has proved extremely instrumental in demonstrating the previously conjectured quasi-instantaneous response of nonlinear crystals when probed off-resonnance. In particular, extremely high gains can be generated in crystals of initially relatively poor quality, owing to the amount of power density resulting from pulse-duration compression at 620 nm of the order of GW's per cm^2. One of the main achievements of this Project has been the demonstration of a new spectroscopic technique termed PASS (after Parametric Amplification and Sampling Spectroscopy) and which may be viewed as anticipating over future high sensitivity, high speed infra-red optical signal processing systems [22,23,7]. It makes use of the unique properties of organic in crystals of the para-nitroaniline family namely : enhanced nonlinearity, adapted transparency in the infra-red and spectrally-non critical phase-matching. The idler frequency at 1.01 μm is prefered as it allows for more sensitive detection and higher signal-to-noise ratio. The advantage of this technique rests on the gain involved as opposed to up-conversion or Kerr spectroscopy with very low yields and on the femtosecond time-resolution (pump duration limited) as opposed to Streak Cameras which are also limited in the infra-red domain.

II.B Molecular systems

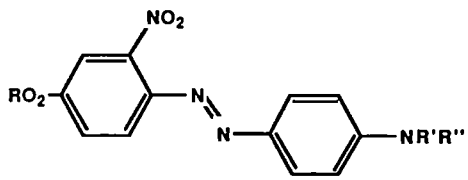
Three directions in organic synthesis have been explored : firstly original molecules which had not been previously considered such as sulphur, metal or other "exotic" atoms containing ones (ICI); secondly, molecules allowing for L-B deposition that are endowed with the proper

hydrophilic/hydrophobic balance to form monolayers at the air-water interface, and with nonlinear properties; thirdly, liquid crystalline polymers capable of hosting nonlinear species either as dopants or grafts in large concentrations and with a T_g significantly above room temperature (Thomson-CSF).

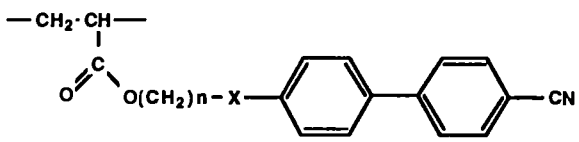
A number of original sulphur containig molecules were synthetized, modelized and measured [24], showing good agreement between theory and experiment, and evidence the dependence of the nonlinearity on the position of the sulphur atom in the conjugated pathway.

Various polyenes have been synthetized for solution testing and L-B deposition. Electron donor and acceptor end groups have been attached to the opposite ends of the molecule and the conjugated chain length has been varied. The nonlinearities of these molecules were measured and shown to reach values as high as 10^{-27} esu. However, these are not yet optimized values owing to the nature of the terminal groups.

A considerable amount of work has been devoted to the preparation of azo compound, for Langmuir-Blodgett deposition. Materials with various chain lengths and donor groups have been prepared to allow for the study of their effect on film forming and packing properties. Most importantly inverse azo compounds have also been made to enable the construction of alternating layers with a significant nonlinear optical response.



Mesomorphous polymers to be used as orientational guest-host matrices were synthesized by Thomson-CSF with para-dimethylaminonitrostilbene (DANS) as the host. Two such families were synthetized : firstly, polyacrylates (I) bearing a cyano-biphenyl side-chain :



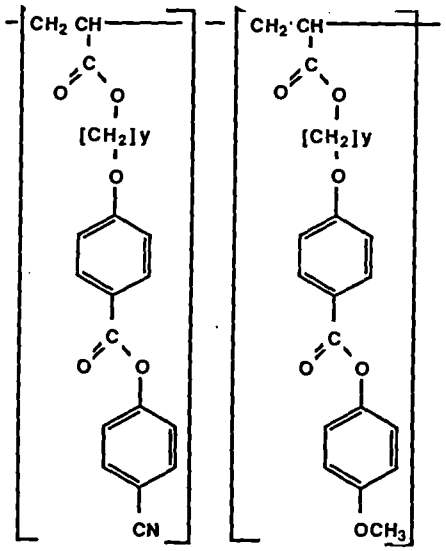
Polymer (I)

With X = COO and n = 5
X = -O- and n = 2 to 6

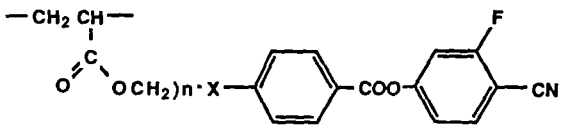
and copolymers with weakly interacting side groups as (II) and (III).

Four polymers with T_g above room temperature and sufficient, however still weak, DANS solubility were obtained.

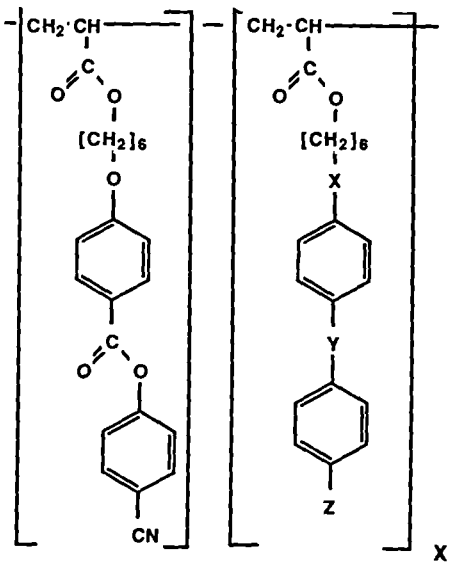
A second approach consisted in the synthesis of copolymers (IV) containing a nonlinear dye as a grafted side group. Efficiencies will be discussed in the next section.



Polymer (II)



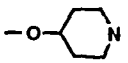
Polymer (III)



Polymer (IV)

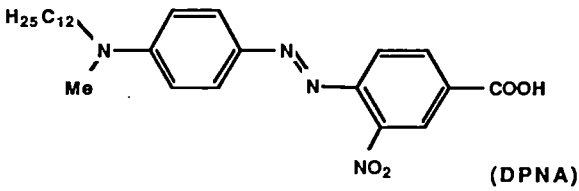
with Z = CN, NO2

Y = -CH=CH-; -N=N-

X = -0-; 

II. C Materials and early devices

A single monolayer of a nonlinear optical dye deposited onto a glass substrate is the simplest reliable method of producing a noncentrosymmetric structure. Hence initial work focused on the examination of a number of compounds deposited as monolayers in order to confirm theoretical predictions of the most active molecules. These molecules included a variety of single ring compounds and azo dyes such as DPNA, and more recently polyenic molecules. In all 18 molecules have been examined.



(DPNA)

A variety of methods exist for the formation of non centrosymmetric L-B multilayers. These are Z and X type films (ie head-to-tail arrangement of the same molecule) and alternating Y type in which alternate layers are either an inactive molecule or an active one in which the polarity is reversed with respect to the structural features of the first active molecule ("active-active" configuration as proposed in Ref. [21]). The architecture of such films is shown in Fig. 5 together with an exemple of molecules specifically synthetized at ICI for that purpose.

Films of all these types have been deposited in multilayer structures up to 20 layers thick, and a comparison of the nonlinear efficiency has been made. It was found that the Y structure containing two active components was the most active and more active than expected from the efficiency of the individual components. Structural effects relating to the substrate film interface have also been examined.

A "first layer" effect was consistently evidenced by various techniques such as pyroelectricity, plasmon coupling and second-harmonic generation [25,26] from an active layer separated from the glass substrate by a variable number of inactive arachidic fatty acid layers. The only structure to display the expected quadratic behaviour with respect to the number of layers is the previously mentionned "active-active" inverted dye Y bilayer structure. Measured $\chi^{(2)}$ values as high as 10^{-6} esu were reached for 11 bilayers at 1.06 μm fundamental wavelength. Waveguiding in sputtered glass-over-glass guides topped by L-B layers was undertaken at CNET. TE and TM mode discretization was observed both at ω and 2 ω. Further experiments of that type using high quality single layers are being undertaken. Light scattering problems on microcrystallites remain to be solved and will obviously depend on identification of structural defects and subsequent adaptation of the deposition conditions.

Oriented nonlinear liquid crystalline films were obtained as reported in Ref.[27]. Using a continuous poling electric field, an oriented guest-host system with 4 % DANS concentration in a type II Polymer (see Section II-B) displays an homeotropic structure (y = 6, T_g = 18.5 °C, T_c = 108 °C). However, the nematic copolymer of type IV (Section II-B) with X : -O-, Y : -N=N-, Z : -C=N, x= 0.24, T_g = 33° C and T_c = 127 °C was poled by a field of the order of 2 V μm^{-1} and an harmonic generation coefficient six fold higher than that of $LiNbO_3$ was measured at 1.06 μm. Further synthetic efforts to raise T_g are under way. Finally, following a technology initiated by Singer et al. [28], dye-doped polymethylmetacrylate (PMMA) glasses were poled and similar SHG efficiencies obtained, i.e. of the order of $LiNbO_3$. An original pathway to overcome orientational relaxation met by the ATT group is being pursued.

The most significant results on bulk single crystals were obtained on N-(4)-nitrophenyl-(L)-prolinol (NPP) [29] shined by femtosecond high power visible laser pulses. Single pass amplification in a sequence of two NPP crystals of respective thicknesses 2 and 1.5 mm and of

unoptimized optical quality showed a gain higher than 10^7 allowing for the detection at 1.1 μm of 1000 photons per 100 femtoseconds.

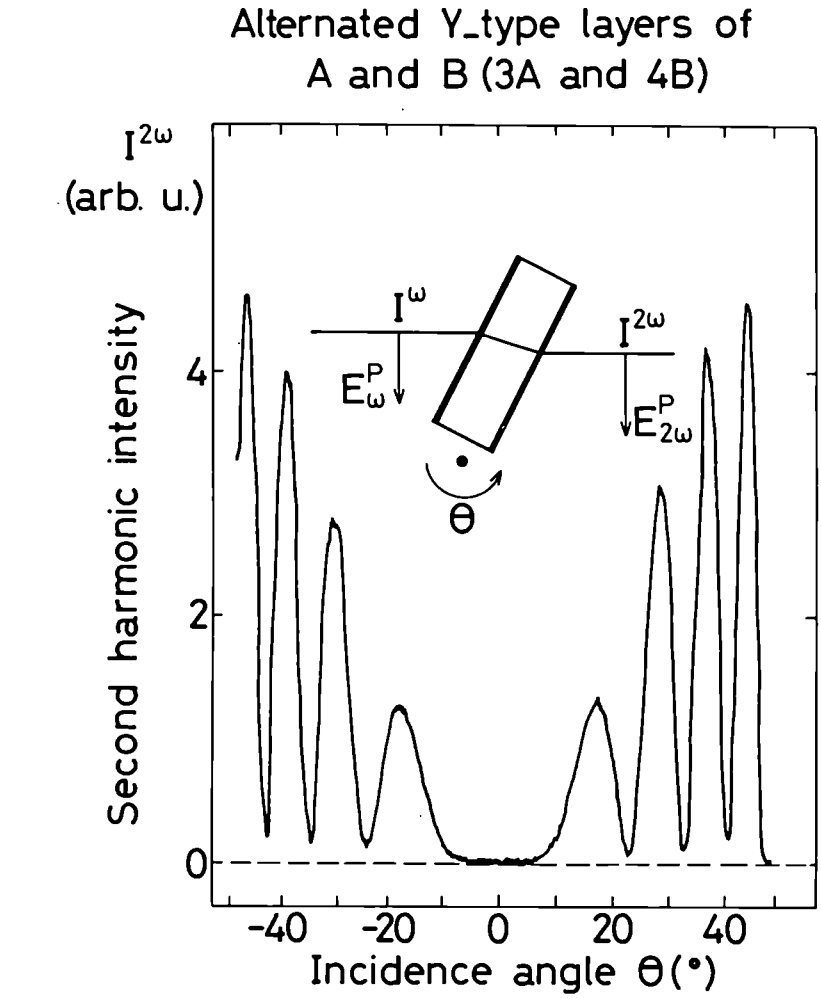


Fig. 5 SHG fringing pattern from a high quality L-B multilayer film resulting from the interference between front and back layers. Varialble dephasing is obtained by rotation of the sample as in Ref. [20]. The film is an "active-active" inverted dye stable Y L-B structure as proposed in Ref. [21].

Figure 6 exemplifies an application to the time-resolution of the luminescence at 1.44 μm of a InGaAs/ InAlAs multiple quantum well (MQW) structure in the previously discussed PASS configuration (Section II-A). Time t=0 corresponds to the arrival of the pump on the sample. A delay of 10 ps, corresponding to the creation and propagation of electron-hole pairs in the p^+-doped 1 μm thick AlInAs buffer layer is noted, the decay corresponding to the trapping and recombination in the GaInAs wells (0.75 eV gap)

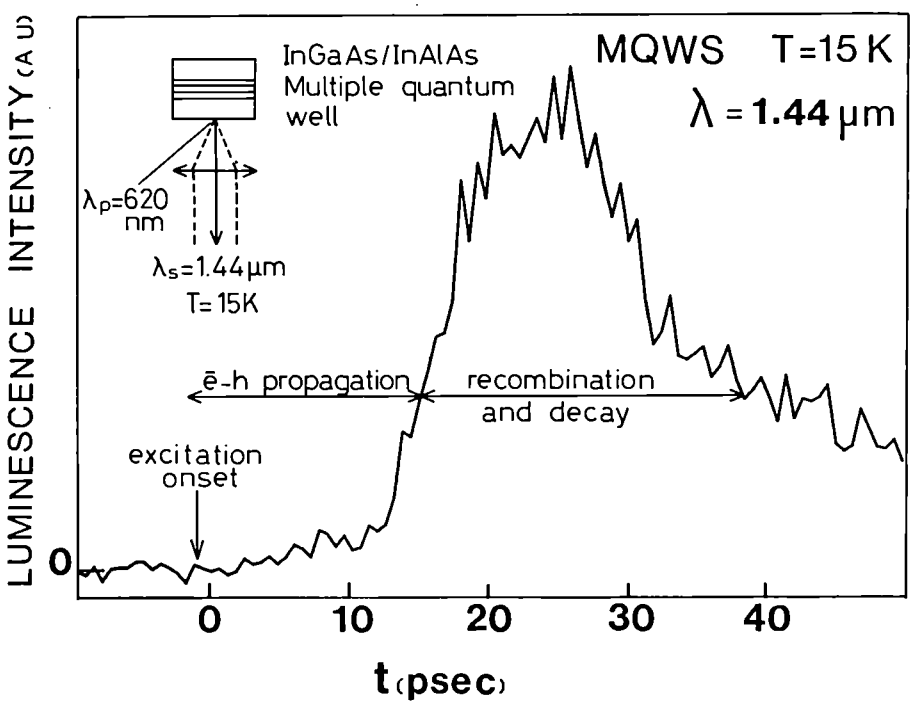


Fig. 6. Subpicosecond resolution of the infra-red luminescence of a InGaAs/InAlAs multiple quantum well structure as from Ref. [23] after excitation at 620 nm by a femtosecond colliding pulse-mode-locked laser.

III Conclusions

The most significant achievements so far in Phase I of this ESPRIT Project are summarized hereafter :

1. the assembly of a reliable programms for the prediciton of molecular β from atomic coordinates;

2. an efficient routine for searching the crystallographic data bases for active molecules of suitable symmetry for $\chi^{(2)}$;

3. an understanding of the effects of hydrogen bonding on hyperpolarizability;

4. methodology to calculate the static polarizability and hyperpolarizabilities of molecules within the Routine Hartree Fock (RHF-Sum Over States and RHF-Finite Field frameworks);

5. methodology to calculate the static polarizability of infinite regular polymers within the RHF-Sum Over States framework;

6. identification of various bonding patterns capable of high electronics responses;

7. synthesis and demonstration of excellent properties in pyrazoline and polyene molecules predicted under 1;

8. setting up of relevant facilities for the characterisation of β and $\chi^{(2)}$ at CNET;

9. the synthesis and dipping of a range of tailor made molecules from which high second harmonic generation has been observed;

10. active molecules have been attached to a polyester backbone to give liquid crystal phases. By aligning under an electric field applied above T_g and cooling to the immobile phase vectorial addition of β has been obtained;

11. the very difficult task of quantifying the factors which govern the lattice symmetry for the polar molecules of interest is now well underway and a number of effective routines have been written to model crystallization;

12. the successes in L-B and liquid crystal techniques are motivated by the desire to construct planar waveguide arrays for integrated optics. L-B technology has highly benefited from Phase I and the knowledge on deposition conditions, structural properties and nonlinear properties has started from almost nil at the onset of the Project to reach a level where technological follow-ups are in view;

13. ultrafast ultrasensitive spectroscopic or I-R signal processing tools (PASS) have been developped.

Building on these significant advances, an ESPRIT Phase II Project could focus on the development of devices towards signal processing or computing whilst maintening the effort on the fundamental substrate on which technology is based. This implies, in the long range, adressing specific, problems such as packaging, electrode deposition, development of device design and fabrication tools, integration of devices onto arrays, piling-up arrays in stacks etc...

In view of the highly challenging goals at stake, one should neither conceal the difficulties nor underestimate the amount of work and time still ahead before devices are being on the market. However significant breakthroughs such as organic bulk Pockels cells, high yield frequency mixers, parametric emitters are possible within the next five years.

The amount of backing which the US and japanese chemical industry is willing to devote to the development of organic nonlinear materials and related devices for IT appears non negligible[30] as compared to that which the electronic industry is investing in III-V compound based nonlinear devices. This situation can be accounted-for by considering that III-V coumpounds may seem relatively more"exotic" as seen from the Silicon foundery than nonlinear organics from a dyestuff manufacturer standpoint.

Therefore, relevant investments in chemistry are marginal while the electronic industry is facing more drastic decisions in order to move into the field of compound semiconductor for NLO.

However, as exemplified by the very composition of the present partnership, it is believed that neither the chemical industry nor the electronic industry alone will be in a position to independently

nurture this new field from infancy onto industrial manufacturing. A cooperative action of the type reported here is in a position to prove its ability to meet the basic problems of this new field.

Further industrial presence of Europe in this highly competitive domain is strongly dependent on the continuation of this effort.

Acknowledgements : Results reported here have result from a cooperation of skills involving over twenty scientists and technicians from three european countries. Working with them has both proved technically and humanely rewarding. Special thanks are due to Dr. R. Murray and Mr. Tainturier with whom enlightening discussions are gratefully acknowledged. Support and encouragement from Dr. O'Shea and Papageorgiou are acknowledged by all.

References

[1] Midwinter, J.E., Light electronics, mith or reality ?, IEE Procedings 132 (1985) 371.
[2] Williams, D.J., Nonlinear Optical Properties of Organic and Polymeric Materials (ACS Symposium Series 233, Washington, 1983).
[3] Chemla, D.S.and Zyss, J., Nonlinear Optical Properties Organic Molecules and Crystals (Academic Press, Orlando, 1987).
[4] Khanarian, G., Molecular and Polymeric Optoelectronics Materials : Fundamental and Applications, Proceeding of SPIE 682 (1986).
[5] Carter, G.and Zyss, J., Nonlinear Optical Processes in Organic Materials, JOSA B 4 (1987).
[6] Professor Kobayashi (University of Tyoko), private communication, in addition to organic nonlinear optics and supraconductors, four Projects are devoted to biotechnology one to III-V compound quantum electronic device and one to biomimetic materials with obvious connections with L-B technology.
[7] Zyss, J., New Nonlinear Organic Crystals for Ultrafast Infra-Red Optical Processing, to be published in: Prasad, P., (ed.), Nonlinear Optical and Electroactive Polymers (Plenum, 1987).
[8] ESPRIT 443 participation to this meeting :
Ledoux, I., Josse, D., Zyss, J., McLean, T., Gordon, P.F. and Allen, S., Second Harmonic Generation in Alternated LB fims; Ledoux, I., Zyss, J., Migus, A., Hulin, D. and Antonetti, A., Processing of Femtosecond Near Infrared Pulses using Nonlinear Crystals; Barzoukas, M., Fremanx, P., Josse, D., Kajzar, F., Ledoux, I., Messier, J., and Zyss, J., Quadratic and Cubic Nonlinearities of Organic Molecules in Solution : New Advances.
[9] Zyss, J., Molecular Engineering for Optoelectronics in: Proceedings of the ESPRIT Technical week '85 (North Holland, Elsevier, 1986).
[10] André, J.M., Morley, J.O., Zyss, J., From Quantum Chemistry to Organic Optical Signal Processing : A computer Aided Molecular Engineering Approach, to appear in: Maruani, A., (ed.), Molecules in Physics, Chemistry and Biology (Reidel, 1987).
[11] Kitaigorodsky, A.I., Molecular Crystals and Molecules (Academic Pres, Orlando, 1973).
[12] Allen, S., ESPRIT Project 443 : Molecular Engineering for Optoelectronic, to appear in: Proceedings of the ESPRIT Technical week '86 (North Holland, 1987).
[13] Morley, J.O., Dougherty, V.J., and Pugh, D., to appear in J. Chem. Soc., Perkin Trans. II and references therein.
[14] Fripiat, J.G., Barbier, C., Bodart, V.P., André, J.M., J. Comp. Chem. 7 (1986) 756.
[15] Barbier, C., to appear in Chem. Phys. Lett.
[16] CNET patents.
[17] Pockrand, I., Sualen, J., Gordon, J., and Philipott, M., Surf. Sci. 74 (1978) 237.
[18] Cross, G.H., Girling, I.R., Peterson, I.R., and Cade, N.A., Electron. Lett. 21 (1986) 1111.
[19] Robin, P. and McLean, T., submitted
[20] Kajzar, F., Messier, J., Zyss, J. and Ledoux, I., Opt. Commun. 45 (1983) 133.
[21] Zyss, J., J. Mol. Electr. 1 (1985) 25, see Fig.2(c) therein.
[22] Ledoux, I., Zyss, J., Migus, A., Etchepare, J., Grillon, G., and Antonetti, A., Appl. Phys. Lett. 48 (1986) 1564.
]23] Hulin, D., Migus, A., Antonetti, A., Ledoux, I., Badan, J., Oudar, J.L., and Zyss, J., Appl. Phys. Lett. 49 (1986) 761.
[24] Gordon, P., (ICI) and Barzoukas, M., et al. (CNET), unpublished.

[25] Ledoux, I., Josses, J., Vidakovic, P., Zyss, J., Hann, R., Gordon, P.F., Bothwell, B.D., Gupta, S.K., Allen, S., Robin, P., Chastaing, E., and Dubois, J.C., Europhys. Lett. 3 (1987) 803.
[26] Ledoux, I., et al. to be published in Thin Film Solids.
[27] Le Barny, P., Ravaux, G., Dubois, J.C., Parneix, J.P., Njeuno, R., Legrand C., and Levelut, A.H., p56 in Ref. [4].
[28] Singer, K.D., Lalama, S.L., Sohn, J.E., and Small, R.D., in Ref. [3] and Small, R.D., Singer, K.D., Sohn, J.E., Kuzyk, M.G., and Lalama, S.J., in Ref. [4].
[29] Zyss, J., Nicoud, J.F., and Coquillay, M., J. Chem. Phys., 81 (1984) 4160.
[30] Professor Peyghanbaryan, Optical Science Center, University of Arizona, private communication.

Project No. 991

OPTIMISATION STEPS IN SILICON COMPILATION

*J.A.G. Jess, J.F.M Theeuwen, R. v.d. Born, L. Stok, M. Berkelaar.**

Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, The Netherlands
Tel. 040-473353
Telex: 51163
UNIX mail: MCVAX!eutes1!lia

ABSTRACT

This paper presents a system capable of automatically generating circuits from algorithmic level descriptions. The algorithm is converted to a data flow graph, which reflects the behaviour of the algorithm and the constraints put in the design. This flow graph serves as a basis for the structural synthesis. A structural synthesis method by means of dynamic programming is presented. The resulting data path and controller are mapped onto transistor networks. The system is able to serve various layout styles.

1. INTRODUCTION

The research reported here has been conducted within project number 991 ("Multiview Design System ICD") together with the partners "British Telecom", (BT) "Perifere Computer Systeme" (PCS), ICS and Delft University of Technology. The primary objective of this project is to support microelectronics technology by providing a modern interactive, modular, open design system for the design of VLSI circuits. The system is to reside in a network of modern desk top workstations possibly supported by computational servers with parallel architectures. The partners have agreed on UNIX and C as primary components of the programming environment enhanced by LISP and possibly PROLOG wherever appropriate. British Telecom, Delft and Eindhoven University are contributing the essential software modules of the system. ICS has started to penetrate the market with a commercial product called SPIRIT based on software modules gathered from BT, Delft and Eindhoven. PCS works on a workstation which is supposed to host all the partners' prototypes as well as the commercial product. The available software however has been ported to many other commercially available workstations.

BT's contribution to the design system is the ASTRA-system, a layout design system with a large amount of automatic routines as parts of it. The contribution of Eindhoven University (which is the specific subject of this paper) comprises a complete set of program modules for the design of system architectures in terms of finite state machines, the design of Boolean logic networks and the design of cells in various layout design styles including *soft macros* based on gate matrix style and *sea of gate*-type gate arrays. It is the intention to combine both systems. First actions have been taken to include the modules for Boolean logic into ASTRA as well as to port ASTRA to the partners and include it into SPIRIT.

Various modules of the system have been applied to a number of designs in the range up to 100 thousand components. This goes in particular for ASTRA which served as a workhorse for a number of commercial designs. However numerous university projects have been supported by modules developed in this project ranging from a fourty thousand transistor chip developed at Delft University for pixel processing down to small training projects aiming at cell designs from a few hundred up to a few thousand transistors.

* This research is sponsored by the European Community under contract ESPRIT991.

Design activity will be enhanced during the last eighteen months of EC-support and thereafter. Eindhoven University will be engaged into two design projects that will yield designs in the range above 100 thousand transistors. One of these projects (which both have a benchmark character rather than being commercial products) concerns the design of a processor with a new type of floating point arithmetic. The other design concerns memories with algorithmic error correction. BT and Delft University are engaged in the design of large chips in the signal processing area.

2. SYSTEM OVERVIEW.

The Eindhoven design system concentrates on the development of a set of program modules crucial for certain optimisation tasks in VLSI-design. These modules are integrated in a complete interactive design system comprising components from all partners. Database manager and design entries are mostly from Delft and Eindhoven. More and more the university prototypes are replaced by commercial ICS modules.

Figure 1 attempts to give an overview over the architecture of the complete design system (verification tools omitted!). According to the principle of top down design we assume a specification of a chip to be documented in a machine readable form. The specification consists of parts dealing with algorithms and performance requirements. We assume that the specification concentrates on describing the function that the chip is supposed to perform rather than the structure. In fact we want to be free to optimise the final chip structure and to ignore the information concerning the structure as far as it is part of the original specification.

From the specification the design proceeds through a number of levels of documentation. The final result is a layout description of the complete chip in a standardised machine readable form. Depending on the design philosophy there may be various intermediate levels of documentation. Completed documents on any of these levels characterise the state of the design. Our convention is to consider two intermediate levels as main levels :

- a description of the chip in terms of *communicating* (or according to Kurshan [Kurshan87] *co-ordinating*) finite state machines (close to the so-called *register transfer level*);
- a description as a network consisting of *modules* or *components* and connections between these components such as wires, buses and the like.

The description in terms of finite state machines can often be decomposed into *data path* and *control path*. If appropriate the designer may stick to such a decomposition. It is often enforced by module libraries supplied by vendors or other third parties. Our own idea, however, is to rather not decompose but consider data and control operations simultaneously during various optimisation steps. We conjecture that, as custom design urges designers to go to the limits of performance of some technology any premature decomposition induced by library modules may preclude optimum performance. Therefore we keep in mind that advanced designers would like to use programs for architecture or Boolean optimisation for the design of the arithmetic parts of the data operations.

Our current high level design entry is an abstract syntax tree format based on LISP-syntax. As standards for function description languages emerge our idea is to provide parsers (by f.i. using LEX and YACC) to map the respective documentations onto our standard format. The advantage of this approach is that within the project we are able to concentrate on the semantic items that are essential for the design and don't lose time with extensive discussions on syntax issues. Moreover this way we keep the system open with respect to later conventions concerning the documentation.

The syntax tree format is converted into a data flow graph called the *demand graph*. The demand graph is scanned for extensive data flow analysis such as for instance the "lifespan" analysis of any value. Also optimisation is performed with techniques borrowed from compilers.

The optimised demand graph is input of a program called *hardware generator*, which establishes the system structure in terms of co-ordinating finite state machines. This step requires the decomposition into library items like adders, multiplexers, bus connections and the like. The decomposition is already visible in the demand graph as the vertices of the demand graph are associated with functions or operations as semantic items. The hardware generator cannot but manipulate those semantic items. However, at this level of design we don't associate fixed layouts with those items. Rather in order to evaluate a design we refer to attributes like *area*, *power dissipation*, *delay* and *complexity*. An item called *ALU* or *MUX* assumes very different forms in the final layout depending on the function it actually performs. The optimisation will go through various assignments of functions to library items as it proceeds. Essential layout features like aspect ratio and pin connections are left open at this stage. They will be filled in by the layout

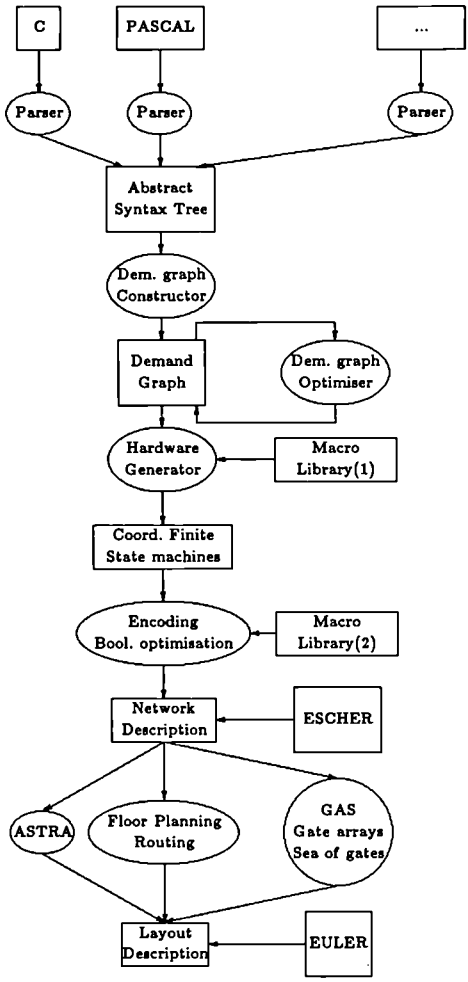


Figure 1. The hardware synthesis system.

generation tools at the bottom of the scheme in Fig. 1.

From the finite state machine description we enter encoding and Boolean optimisation in order to generate a network description of the chip to be designed. Usually the finite state machine description will still be decomposed into parts which are linked to semantic items of the designer's world. Straightforward Boolean optimisation may proceed by optimising any part separately. However other methods can be thought of. For instance the whole combinational Boolean system between two state latches could be considered as one chunk for the Boolean optimiser. If diligently done this enhances the scope for optimisation considerably. As exemplified by the benchmarks given in Chapter 5, our Boolean optimisation tools are designed to cope with Boolean functions of the appropriate size. We continue down to the bottom of Fig. 1 by entering various layout construction modules. Our main goal is to support flexible layout styles like *gate matrix* or *sea of gate* images. It is well known that floorplanning works best with cells that are flexible with respect to pin connections and aspect ratio. Recently it has been suggested that the limits of performance

mandatory for custom design can only be achieved by *soft macro* design methods [Koetzle87]. Our system is developed to support the soft macro approach in all phases of design.

3. DEMAND GRAPH GENERATION.

The demand graph generator currently accepts the following constructs:

— assignments;

— operations like *, -, +, ..;

— branch constructs (**if .. then ... else, case ... of**);

— loop constructs (**while ... do, for ... do**);

— port read and write operations (**put, get**);

— operations to access bit fields in variables;

— array access and update operations;

— procedures and functions in order to deal with hierarchy and recursion;

— structures for describing asynchronous communication (**message.., wait**);

To demonstrate the demand graph construction we use a design example reported by Tseng [Tseng83]. Tseng's design example is an algorithm to be mapped onto silicon. The description of the algorithm in a LISP like syntax tree notation (our current input medium) is given in Fig. 2 . The demand graph constructor scans the syntax tree and attaches nodes according to the demand graph. The resulting demand graph is shown in Fig. 2 .

```
(program Tseng ()
      (v1 v2 v3 v4 v5 v6 v7 v8 v9
        v10 v11 v12 v13 v14 v15)
      (; (get v10 v6 v4 v2 v1)
      (:= v3 (+ v1 v2))
      (:= v5 (- v3 v4))
      (:= v7 (* v3 v6))
      (:= v8 (+ v3 v5))
      (:= v9 (+ v1 v7))
      (:= v11 (/ v10 v5))
      (:= v12 100)
      (:= v13 v3)
      (:= v12 v1)
      (:= v14 (and v11 v8))
      (:= v15 (or v12 v9))
      (:= v1 v14)
      (:= v2 v15)
      (put v1 v2)
      ))
```

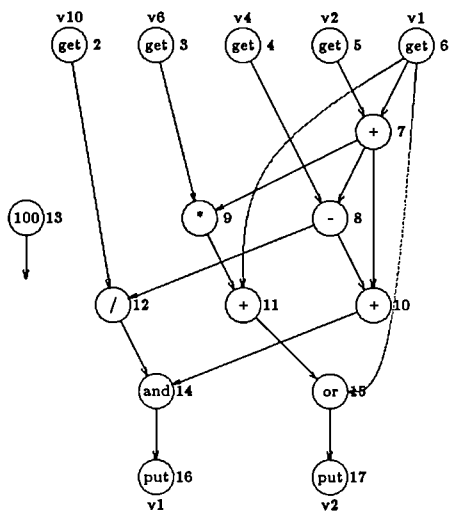


Figure 2. Algorithm and demand graph for Tseng-example.

In the figure we distinguish various types of nodes:

— round nodes indicating operations +, * -, /;

— the *get*- and *put*-nodes take care of the data transfer from and to the outside world.

— lines with arrows (*arcs*) indicate the data flow through the algorithm.

The demand graph is acyclic which follows from the absence of loop constructs in this simple example.

The *demand graph optimiser* scans the demand graph in order to perform the following optimisations:

— *Redundant subexpression elimination*: if two operators that both compute the expression A * B are separated by code that contains no store into either A or B, then the second operator can be eliminated, if the result of the first is saved.

— *Constant folding*: if all inputs to an operator are constants which values are known, the result of the operator can be computed at compile time and stored instead of the operator.

— *Code motion*: Operators that depend upon variables which values do not change in a loop may be moved out of the loop.

— *Strength reduction*: operators that depend on the loop induction variable cannot be moved out of the loop, but sometimes they can be replaced by less expensive operators.

— *Variable folding*: statements of the form A : = B will become useless if B can be substituted for subsequent uses of A.

— *Dead code elimination*: if transformations like variable folding are successful, there will be many operators whose results are never used. Dead code elimination detects and deletes such operators.

— *Procedure integration*: under certain circumstances a procedure call will be replaced by the body of the procedure being called.

In the simple example of figure 2, the only optimisation carried out is the elimination of node 13, because it does not contribute in any way to the output.

4. SYNTHESIS OF AN OPTIMAL STRUCTURE.

Generating an efficient implementation of a demand graph requires the following interdependent tasks to be performed:

— A network of registers, operators, multiplexers, and controllers has to be constructed.

— Instructions (nodes) are to be mapped to operators and machine cycles.

— Edges are to be mapped to busses and if the adjacent instructions are assigned to different machine cycles, to registers

The demand graph could be mapped onto silicon node by node. Simultaneously the appropriate controlling finite state machine could be developed easily. This way, however area intensive solutions would be obtained. Consider for instance the case of a demand graph with ten nodes performing addition. A straight forward mapping of the demand graph in silicon would include the area for ten different adding devices. Of course a more area economic solution would contain only one or two adders. This implies the following actions:

— extra wire connections and multiplexers must be provided to rout the arguments to the adding device in question and to rout the output to where it belongs.

— a schedule must be developed that enforces correct use of the adder by preventing the interference between different additions on the adding device. This may imply delays but also extra registers to buffer arguments and results.

Many different options are possible and the best feasible is to be found.

The method proposed here, unlike other methods with similar objectives, considers all optimisation problems simultaneously and is therefore able to take their dependency into account.

4.1 Partial nodesets

To explain how the demand graph is traversed the following notions are used:

— a *partial nodeset* is a set of nodes of the demand graph, such that if a node is member then all its predecessors are.

— a node is said to be *free* with respect to partial nodeset *S* if the node is not in S but all its predecessors are.

— the set of *free nodes* of a partial nodeset S is the set of all nodes that are free w.r.t. S.

Set inclusion establishes a partial order over the set of partial nodesets. Since the empty set and the set of all nodes are members of the set of partial nodesets, the partial nodesets form a lattice: the *nodeset-lattice*. As an example the nodeset-lattice associated with the demand graph in Fig. 2 is given in Fig. 3 . In the figure the numbers in the lattice vertices correspond to the demand graph node numbers of Fig. 1 and represent the set of free nodes of that lattice vertex.

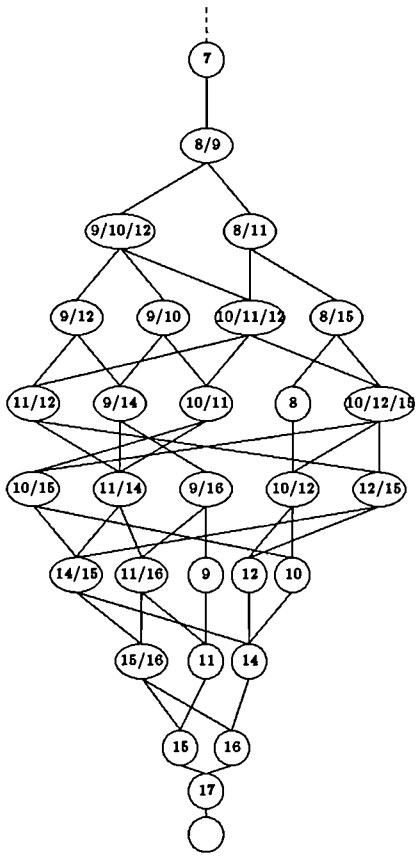


Figure 3. Nodeset-lattice for the Tseng algorithm.

The lattice is strictly leveled. That is the level of a partial nodeset is determined by the number of nodes in that partial nodeset. Furthermore the lattice can be easily generated using induction:

— the empty set is a partial nodeset.

— if N is free w.r.t. the partial nodeset S then S ∪ {N} is a partial nodeset.

4.2 Structure synthesis

In its most primitive form the generation of implementations is done by going over the nodeset-lattice level by level. For each partial nodeset a *partial implementation* is maintained. While generating a new level a free node is "implemented" on top of the partial implementations of the old level. If more partial implementations for a single nodeset are obtained the least expensive (with respect to some cost-function) is chosen.

This approach proves to be too simple and discards valuable solutions prematurely. Therefore the method is extended as follows:
More partial implementations for the same nodeset are allowed to coexist. In order to decide which implementations are kept we introduce the notion of *comparability*. Two implementations are *comparable* if they satisfy a relation defined by a boolean predicate. The user is basically free to define this predicate according to his own insights. Usually a necessary condition for two implementations to be comparable is that they implement the same nodeset. For our design example we considered two implementations comparable if and only if they implement the same nodeset and their respective numbers of so called *large* modules (like adders or ALU's) are equal. Among the pairwise comparable implementations we maintain the cheapest in terms of the cost function.

The algorithm performing the optimisation is shown in Fig. 4 .

```
next_level := initial_level();
repeat
    level      := next_level;
    next_level := empty;
    for old_impl in level do
        for node in free_nodes( old_impl ) do
            generate implementations for node given old_impl;
            for i in these implementations do
                if i is comparable to one from next_level
                then
                    select cheapest for next_level;
                else
                    add to next_level;
            done;
        done;
    done;
until empty( next_level );
```

Figure 4. Algorithm for synthesis

As can be seen a breadthfirst search for implementations is effectuated discarding overexpensive solutions on its way. This is essentially a dynamic programming method, a well known optimisation method [Bell62].

As a cost function we use:

$$Cost_{implementation} = area * (delay * \#cycles).$$

The product of *delay* and the number of *cycles* (= number of states of the state machine) gives an indication of the time used by the implementation. The product of this with the *area* occupied by the implementation gives us a volume in the design space. This volume can be used to choose the best of this implementation and other *comparable* implementations.

Assuming we choose for an cost function the function given above we obtain the structure of Fig. 5, controlled by the finite state machine given in Table 1, as a result of aplying the algorithm in Fig. 4 to Tseng's design example.

TABLE 1. Controlling finite state machine generated for Implementation_1.

State	Inputs	Next-state	Outputs										
			Muxes						Reg enables				
#0	none	#1	0	0	0	0	0	0	1	1	1	1	1
#1	none	#2	1	1	1	1	1	1	0	0	0	0	0
#2	none	#2	x	x	x	x	x	x	0	0	0	0	0

Table 2 shows the flexibility of the method. Several different implementations of Tseng's design example are generated by merely changing bounds or initial conditions. The constraints contain *maximum area, maximum delay* and *maximum number of cycles* that may be used by an implementation. With delay we mean the duration of one cycle of the state machine. Row three of Table 2 describes an implemenation (Impl-3) with no bounds on the area and the delay (*large*) and a maximum of one cycle. The system will deliver a fully combinational circuit with no registers and multiplexers and just as many operator modules (Add, ALU etc.) as operations are in the demand graph. Impl-4 may use maximal 8 cycles and the delay allowed is the duration of a multiplication (=

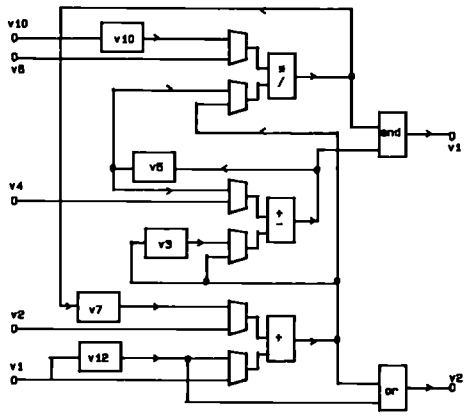


Figure 5. Implementation_1 for the Tseng-algorithm

mult). This results in an implementation with 4 cycles, one multiplier and several registers, multiplexers and small operator modules.

TABLE 2. Implementation for the Tseng algorithm with different bounds.

Tseng example											
	constraints			results							remarks
	area	delay	#cyc	#cyc	#ALU	#Mult	#Add	#And/Or	#Reg	#Mux	
Impl-1	large	< 2 mult	6	2	-	1	2	2	5	6	
Impl-2	small	large	8	3	-	1	1	2	5	7	
Impl-3	large	large	1	1	-	2	4	2	-	-	
Impl-4	large	= mult	8	4	-	1	2	2	5	10	
Impl-5	large	< 2 alu	6	4	3	-	-	-	5	7	ALU's only

5. *SYNTHESIS OF COMBINATIONAL CIRCUITS.*

We now want to convert the descriptions, obtained by the structural synthesis, into actual networks. These may be *Boolean* which is to say they are in terms of elementary Boolean functions or gates. In the technology of integrated circuits, however, we often have to go down to the level of transistors, for instance if pass transistor logic is involved. Moreover delay and area predictions are more dependable on the level of transistors.

State variables, input and output variables have to be latched. Latches may be considered to be standard circuits. According to the structure of the machine there are combinational logic functions between those latches. The essential problem of logic synthesis is to map the combinational logic functions onto transistor structures in an optimal way.

It has been observed long ago that the form of the combinational logic functions depends largely on the binary encoding of the states, inputs and outputs [Hart66]. Some approaches have been published trying to tackle the problem of finding optimum encodings [Hart66], [Mich85]. Our own experiments suggest, however, that more research on this problem has to be done.

Let us thus assume that state, input and output encodings have been fixed. In that case the Boolean functions of a system are defined by their so-called ON-set, DC-(don't care)set and OFF-set. These are sets of minterms in the Boolean input space. Their meaning is self-explanatory. For any function the union of the three sets is the complete Boolean input space.

Practically all combinational logic functions are defined by sums of *cubes* or *implicants*, which are products of Boolean variables. Those products denote Boolean subspaces of the input space. Such a representation is usually more compact as compared to representations by minterms (so-called *truth tables*). In our current setting the synthesis of combinational logic functions develops along four steps:

1. Two level minimisation;
2. Multilevel logic minimisation;
3. Technology mapping;
4. Delay optimisation.

Two level minimisation attempts to minimise the number of implicants and in addition the overall number of literals (one term factors) in the implicants. Programs as *ESPRESSO* in its various versions [Bray84], *McBoole* [Dage86] and some others are the most recent offsprings of a long history of research in this area.

5.1 The multilevel logic optimiser

Multilevel logic minimisation takes the output of a *two level minimiser* as input. The goal is to find common subexpressions in various different Boolean functions. If such a subexpression is made to define a new variable and this variable is substituted for the subexpression, then the number of transistors is reduced at the expense of introducing additional delay.
Our approach follows the approach of Brayton a.o. [Bray84]. First we systematically search for *kernels*. Kernels are subexpressions satisfying the following conditions:

— they are a sum of at least two implicants;

— none of these implicants has a cube (subimplicant) in common with another implicant in the sum.

We make the acceptance of a kernel for substitution dependent on two attributes:

— the number of implicants in the kernel, the *kernel size*;

— the number of times the kernel appears, the *kernel count*.

We specify *minimum-kernel-size* and *minimum-kernel-count* and substitute the kernel every time both its attributes meet at least the minimum specifications.
After substituting kernels we look for common cubes (as the complements of cubes are kernels as well). We substitute them in much the same way depending on the attributes *minimum-cube-size* (number of literals in a cube) and *minimum-cube-count* (self explanatory!).
Our program allows for the formal definition of a delay model per Boolean expression. For any individual Boolean expression the delay model can be an almost arbitrary function of parameters like fan-in, fan-out, active gate area, complexity of the Boolean expression and the like. The program monitors the critical paths through the multilevel network by evaluating those delay models. Kernels and cubes can also be rejected if a predefined critical path length would be exceeded by the substitution.

5.2 Technology mapping.

We simplify this generally very complicated problem by defining a standard function namely the *n-cube n-literal and-or-invert* (*n*-AOI) function:

$$f_{\text{n-AOI}} = \text{NOT}\left(\sum_{i=1}^{n} X_{i,1} \cdot X_{i,2} \cdot \ldots \cdot X_{i,n}\right)$$

Note that this function has the form of a kernel or the complement of a cube. The number of implicants and of literals per implicant must be no greater than n in $f_{\text{n-AOI}}$. The function is readily available in very stable static NMOS and CMOS circuits. The value of n is considered to be given as a technology constraint.
All functions emerging from the multilevel logic minimiser are checked on fitting onto a n-AOI function. If not they are split until they fit. Critical path length is monitored, but as n is fixed nothing can be done if the delay limit is exceeded.

5.3 Delay optimisation.

Our splitting process in *technology mapping* usually introduces excess inverters, which, however, can only be recognised to be unnecessary as the whole network is screened. Thus a heurisitic procedure has been developed that manipulates inverters in such a way that the delay along the critical path is minimised. During this minimisation the delay along the critical path is evaluated according to the delay formulas as defined by the user. If in accordance with those delay models the elimination of inverters yields a speed-up then such elimination is effectuated. This process is supported by the fact that the restrictions to the n-AOI-functions inversion only requires

application of De Morgan's law rather than formal Boolean inversion. Within our system experiments show the delay optimisation to provide significant improvements for little computational effort as usually many inverters are removed.

5.4 Results

The system described above has been used in a number of designs which, however, did not exploit the full power of the algorithms. Therefore we have tested the system with two sets of benchmarks, that circulate internationally among the research institutions and reflect a mix of requirements that were derived from actual design situations. Some of these are of considerable complexity. Others are designed to exhibit certain typical weaknesses of programs of this kind. As a number of subproblems of Boolean optimisation are nondeterministic polynomial complete no finite set of polynomial heuristics can cover the optimisation problem. That is for any polynomial heuristic algorithm there must be a benchmark demonstrating either exponential time or exponential memory claims. Our system has been engineered to the extent that all benchmarks mentioned here have been successfully completed on a desktop workstation.

Table 3 shows the results of a set of benchmarks taken from [Bray84]. We consider the inputs such as they emerge from ESPRESSO, that is after two-level minimisation. For any circuit we give three results.

In the first result we bypass the multilevel logic minimiser and run the ESPRESSO output directly through the *technology mapper* and the *delay optimiser*. For the second result we fix the possible parameters (minimum-kernel-size, minimum-kernel-count, minimum-cube-size, minimum-cube-count) all to the value *3*. For the third set of results all these values are set to *2*. Note that the smaller we fix those parameters the more freedom the multilevel minimiser obtains. The results show the following general tendency:

— transistor counts are drastically reduced by the action of the multilevel logic minimiser;

— delay does not develop uniformly. With no multilevel minimisation delay still is small due to a substantial amount of parallelism in the circuit. Although choice *3* for the parameters reduces the average transistor count considerably, we see that choice *2* not only reduces the transistor count even further but in addition improves the delay (on the average).

The last observation is probably related to the fact that we use 3-AOI circuits throughout. If we fix the *size* parameters to *3* too many large subcircuits stay intact. It is obviously less effective to let them be split by the *technology mapper* instead of letting the *multilevel logic minimiser* do the job. Setting the parameters according to the pattern *2-4-2-4* gives a slightly better average delay result (7.17) at the expense of a slightly higher average transistor count (172).

Table 4 shows an alternate set of benchmarks that have been run with several synthesis systems, among others the MIS-system of the University of California in Berkeley. All results have been presented at the *International Workshop on Logic Synthesis* held at Research Triangle Park, North Carolina, USA from May 12 to May 15, 1987. Our system was able to complete almost all benchmarks successfully within reasonable time. A comparison of our results with those of other systems is difficult as the systems distinguish in various details. For instance technology mapping is tuned to different libraries of Boolean function cells. However in a global sense our results were competitive. With the benchmarks unit delays per Boolean expression have been used because no information about more elaborate delay models came with them.

6. CONCLUSIONS.

We have presented algorithms and results concerning the high level phases of the design of digital systems. The first part of the paper concerns the synthesis of structures consisting of large modules. This part (called *hardware synthesis*) attempts to sketch a systematic approach to synthesis showing that various substantially different approaches fit into one algorithmic scheme.

It also shows results from a prototype implementation that has been applied with some simpler designs. This implementation is currently in the process of further development and will be used to optimise the architecture of various large demonstrator chips.

The second part considers the subject of optimising combinational Boolean functions. It shows results of a system which is still in the process of improvement. The essential objective is to broaden the scope of optimisation towards more different technologies.

Also this system has been applied in various design situations. The verification of viability is attempted by using benchmarks circulating internationally.

TABLE 3. Results of logic synthesis; technology NMOS; standard circuit 3-AOI; any load counted as one transistor.

	no multilevel minimisation			all parameters set to 3			all parameters set to 2		
circuit	# trans.	# cells	delay	# trans.	# cells	delay	# trans.	# cells	delay
alu1	49	8	2	49	8	2	49	8	2
alu2	538	8	11	399	48	10	211	41	8
alu3	298	8	7	252	24	8	160	33	7
apla	797	12	6	321	45	12	288	61	12
col4	197	1	8	197	1	8	113	11	12
dc1	78	7	4	68	12	4	68	17	6
dc2	267	7	5	209	27	10	177	38	10
dk17	400	11	6	185	27	10	179	40	8
dk27	189	9	6	142	21	6	116	28	7
in6	767	23	9	441	63	12	424	87	9
in7	562	10	9	235	29	10	202	44	9
misg	203	23	8	189	30	12	152	36	10
mish	122	21	4	122	21	4	114	28	4
radd	345	5	6	233	19	11	125	29	8
rd53	159	3	6	139	15	8	79	15	6
risc	233	31	4	170	45	6	178	55	6
sqn	237	3	5	207	22	9	184	31	11
wim	74	7	4	68	9	4	68	18	6
average	306	11	6.06	201	26	8.11	160	34	7.83

TABLE 4. Results of the *Research Triangle Park workshop* Benchmarks.

LIBRARY: 4-AOI						
	no multilevel minimisation			all decomp. parameters 2		
circuit	# trans	# cells	delay	# trans	# cells	delay
5xp1	228	57	4	191	62	5
9sym	303	54	6	330	120	8
bw	521	102	4	258	83	5
con1	26	9	3	23	8	2
duke2	1166	309	6	593	243	9
f2	28	8	2	20	8	2
misex1	82	23	5	67	14	2
misex2	580	182	6	317	130	9
rd53	87	21	4	54	17	4
rd73	301	61	6	153	53	6
rd84	542	104	7	263	95	6
sao2	290	71	6	269	102	9
vg2	336	85	5	151	64	6
average:	345	83	4.9	206	76	5.6

As of now the systems support each other. Furthermore the logic optimisation is linked to NMOS and CMOS layout generators that map the logical nAOI-network into silicon in standard cell style. The system also links to a gate array *place and route*-system which can handle *sea of gates*-type array images. Various cell generating schemes based on the *gate matrix* approach are operational.

REFERENCES

[Bell62] Bellman, R.E., and Dreyfus, S.E., *Applied dynamic programming*, Princeton: Princeton University Press, 1962.

[Bray84] Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni Vincentelli, A., *ESPRESSO-II: Logic minimisation algorithms for VLSI Synthesis*, The Hague: Kluwer Academ.Publ., 1984.

[Dage86] Dagenais, M.R., Agarwal, V.K., Rumin, N.C., "McBoole: a new approach for exact logic minimisation", *IEEE Trans. Comp. Aided Design of Circuits and Systems*, vol.CAD-5, pp. 229-238, January 1986.

[Denyer86] Denyer, P.D., "Silicon Compilation.", in: *Proceedings of the ESSCIRC'86*, Delft, 1986, pp .34-37.

[Gee85] Gee-Gwo, M., Wentai, L., "Data path synthesis with/without constraints." *Proceedings of the ICCAD85: 3rd conference on computer aided design*, Santa Clara, 18-21 Nov.1985.

[Hart66] Hartmanis, J. and Stearns, R.E., *Algebraic structure theory of sequential machines*, Englewood Cliffs: Prentice Hall, 1966.

[Jess86] Jess, J.A.G., Slenter, A.G.J., "The prototype of an open design system for gate arrays.", *Esprit'86: Results and achievements*. Elsevier science publishers b.v., 1986, pp.541-550.

[Jess87] Jess, J.A.G., "Synthesis of structures and logic under VLSI conditions.", *Proceedings of Compeuro87*, Hamburg, May 1987, pp. 293-298.

[Kenn81] Kennedy, K., "A survey of data flow analysis techniques", in: S.S. Muchnick, N.D. Jones *"Program flow analysis: theory and applications"*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 5-54, 1981.

[Koetzle87] Koetzle, G., "System implementation on a highly structured VLSI master image", *Proceedings of the Compeuro 87*, Hamburg, pp.604-609, 1987.

[Kurshan87] Kurshan, R.P., Gertner, I., "Logical analysis of digital circuits", *Proceedings of the CHDL 87*, Amsterdam, 1987, pp.47-67.

[Mich85] De Micheli, G., Brayton, R.K., Sangiovanni Vincentelli, A., "Optimal state assignment for finite state machines", *IEEE Trans. Comp. Aided Design of Circuits and Systems*, vol. CAD-4, no. 3, pp. 269-285, July 1985.

[Stok86] Stok, L., *"Higher levels of a silicon compiler."*, EUT Report 86-E-163, Eindhoven University of Technology, November 1986.

[Theeuw85] Theeuwen, J.F.M., van Paassen, P.T.H.M., "Automatic generation of Boolean expressions in NMOS technology", *Proc. Int.Conf. Comp. Aided Design (ICCAD85)*, Santa Clara, 1985, pp.332-334.

[Theeuw87] Theeuwen, J.F.M., Berkelaar, M.R.C.M., "Logic optimisation with technology and delay in mind", *Notes of the International workshop on logic synthesis*, Research Triangle Park, North Carolina, May 12-15, 1987.

[Tseng83] Tseng, C.J., Siewiorek, D.P., "FACET: A procedure for automated synthesis of digital systems.", *Proceedings of the 20th Design Automation Conference*,1983, pp. 490-496.

[Veen85] Veen, A.H., *"The misconstrued semicolon"*, Dissertation, Eindhoven University of Technology, CWI, Amsterdam, 1985.

Project No. 97

SILICON COMPILATION OF DSP SYSTEMS WITH CATHEDRAL II

*H. De Man, J. Rabaey,** J. van Meerbergen, J. Huisken

* *IMEC* ** *Philips Research Labs*

1 Abstract

This paper describes the methodology of the CATHEDRAL-II system, an environment for the efficient synthesis of complex digital signal processing (DSP) circuits. The synthesis process translates a behavioral, flowgraph-type algorithm description, expressed in the SILAGE language [Hil85] into a dedicated multi-processor architecture. CATHEDRAL-II allows the system designer to investigate and compare in an interactive way a number of silicon implementations of a certain DSP algorithm. The application range of CATHEDRAL II is situated in the fields of audio, speech, telecom, lower end video, control and linear matrix operations.

2 Introduction

The rising complexity of the algorithms which can be integrated on a single chip, and the shorter lifetime of the developed products have prompted a large effort towards the development of automated synthesis tools. These systems (also called silicon compilers) try to translate a high level behavioral description of an algorithm into a silicon implementation.

A large number of synthesis techniques and systems have been published recently (e.g. [Kow85], [Mar86]). The majority of these systems however lack the efficiency or the flexibility, needed to span the above mentioned application area. In this paper, we present the CATHEDRAL-II system, which is based on following general principles in order to cope with the mentioned deficiencies.

The major strength of the CATHEDRAL-II system is its clear definition of a *target architecture*. In fact, it is our belief that effective design synthesis is only feasible if the underlying architecture with its operator types, interconnection strategy and memory mechanisms has been defined. This is especially important in DSP applications, where efficiency is of prime importance. The synthesis tools and the optimization criteria are heavily influenced by the choice of the architecture. E.g. the synthesis of bit-serial, hardwired bit-parallel or programmable bit-parallel architectures requires different techniques and procedures. We believe that in the future, a library of Application Specific Silicon Compilers will emerge in order to span the complete ASIC field.

In order to support a large variety of applications, the selected architecture has to be flexible, parameterizable and extendable. The synthesis system has to allow for the user introduction of newly defined hardware units and implementation (or translation) protocols. Therefore, a rule base system has been selected to *describe* the architecture and its protocols and procedures. Changes to the architecture can be introduced through a knowledge acquisition system. For each architecture, also a number of procedural optimization tasks can be defined as critical path computation, scheduling, memory minimization, etc.

An important feature of CATHEDRAL-II is its openness to user interaction. We believe that a 'push-the-button' compilation strategy cannot result in efficient solutions for all cases. Therefore, the designer is allowed to iteratively refine the initial design, proposed by the system. This user interference happens with the aid of a number of high level structural constructs (further called pragma's), which can be considered as constraints on the compilation process. Interactivity also assumes the feedback of enough relevant information (data path structure, area, cycle count, bottlenecks) to the user.

The above mentioned topics will now be discussed in more detail. After a discussion of the selected target architecture, the different system tools will be discussed. The paper will be concluded with a number of examples.

3 Target Architecture ([Catt86], [DeM86])

Extensive studies of a large number of industrial applications have shown that a dedicated processor architecture is best suited for the intended application range. High throughput rates can be obtained by placing multiple processors on the same chip, each of them being optimized to perform one particular part of the algorithm. An example of such a configuration is shown in Fig.1a.

Each of the processors consists of a dedicated data path and controller. The datapath is tailored to the application and consists of a set of Execution Units (EXU's), connected by a number of customized busses (Fig. 1b). The set of the available EXU's is restricted to six as determined by the application studies. They can be divided in two classes : general purpose units as an ALU/Shift and an Address Computation Unit (ACU), and accelerators as a multiplier/accumulator, a divider, a comparator and a normalizer. All these modules have been stored in a parameterizable fashion in a module generator environment [DeM86] : examples of possible parameters are the wordlength, the depth of the shifter or the register files, the optional removal of a unit, the type of the adder used, etc. All these modules have been fully characterized and a variety of views (as functional, area, black box, power, timing and test) are available for use by the synthesis or floorplanning tools.

A powerful microcode-rom based multi-branch controller architecture has been selected though other architectures as e.g. a simple FSM can also be incorporated. A full range of synchronous interprocessor communication protocols have been provided as switched RAM's (Fig. 1c), single and double buffered FIFO's or ready/acknowledge based protocols. A central controller directs the traffic between the processors and to the I/O circuitry.

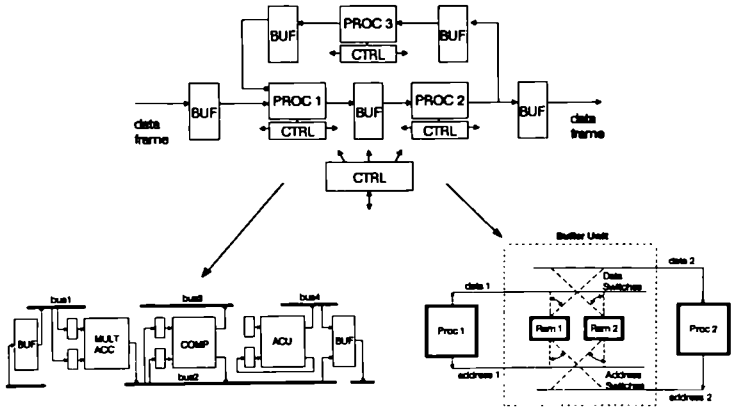


Figure 1: Multiprocessor architecture (a) including internal processor structure (b) and interprocessor communication (c)

4 Synthesis Tools

The synthesis task can now be defined as the translation of the behavioral description of an algorithm into the above defined architecture. Following subtasks can be identified : specification and simulation, processor partitioning, data path synthesis and operator assignment, controller synthesis (including microcode scheduling), selection of the interprocessor communication protocols and generation of the central controller. Tools have been developed (or are under development) for most of the above defined tasks with exception of the processor partitioning, which is performed manually at present. An overview of the total system is given in Figure 2. A number of the mentioned tools and representations will be discussed in more detail.

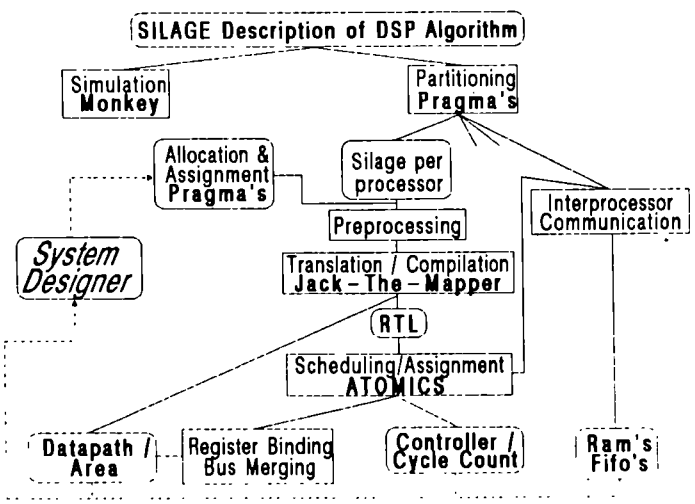


Figure 2: CATHEDRAL-II Synthesis Toolbox

4.1 System Specification and Simulation

We have selected SILAGE [Hil85], a language optimized for the high level description of signal processing algorithms, as the design language for CATHE- DRAL-II. The main idea of SILAGE is to capture the signal flowgraph nature of a signal processing algorithm. It does not contain any structural or control information and does not enforce any degree of concurrency.

The SILAGE description of a PCM Filter can be found in Fig. 3. This example illustrates that SILAGE only contains behavior and does not impose any structure. The structural representation will be generated by the compiler. In the interactive concept of CATHEDRAL-II however, the system designer should be able to enforce structural decisions and this at as high level as possible. This can be done by adding 'pragma' statements to the behavioral description. Pragma's are incomplete structural hints to the compiler.

The SILAGE description of the algorithm can be considered as a system specification and should therefore be debugged and verified in a rigorous way. For the system designer, SILAGE also serves as the algorithm development medium. Hence, efficient simulation tools are of prime importance. A demand driven SILAGE simulator, called MONKEY, has been developed. At present, we are also studying real time SILAGE emulation.

```
#define WORD num<8,0>

#define all 0.625          /* 0.101'      */
#define a12 1              /* 1.0'        */
#define b11 0.5            /* 0.1'        */
/* #define b12-0.375       *//* 0-01'     */
#define b12 0.375          */0.-01'       */
#define a21 7              */100-'        */
#define a 22 1             */1.0'         */
#define b21 0.3125         */0.0101'      */
#define b22 0.78125        */-.0100-'     */
/* #define b22 -0.78125 *//* -.0100-'
#define a31 0.5           /*0.1'          */
#define b31 0.375         /* 0.10-'       */

func main (In:WORD) Out : num =
begin

  Section1 =  biquad (In, all,a12,b11,b12);
  Section2 =  biquad (Section1,a21,A22,b21,b22);
  Out : FirstOrder (Section 2,a31, b31);
end

func biquad (input,a1,a1,a2,b1,b2 : num): num =
begin

  State = input + WORD (b1 * state@1) - WORD (b2 * State @2);
  return = State + WORD (a1 * State @1) + WORD (a2 * State @2);
```

```
end;

func First Order (input,a,b: num) : num =
begin
  State = input + WORD (b *STATE @1);
  Return = State + WORD (a * State @1);
end;
```

Figure 3: Silage description of a pcm filter

4.2 Data Path Synthesis

The synthesizer, JACK-THE-MAPPER, deduces a datapath structure, consisting of a set of EXU's, from the SILAGE description. The methodology, adopted in JACK, is to use a mixture of automated tools and user interaction to solve this extremely complex optimization and search problem. In fact, our experience has shown that a system designer often has a good insight in the complexity and the computational bottlenecks of an algorithm. Therefore, he is well capable to estimate the required amount of parallelism and the acceleration units needed. The most time consuming and error prone job is not located in the allocation task, but in the operator assignment, the controller generation and the minimization of the execution time, the register usage and the bus count. This is where the synthesis task has to come in and has to be extremely good in order to be acceptable.

Based on these considerations, we have deduced following synthesis strategy (Fig. 2) :

- The SILAGE description of the algorithm is first passed to a *preprocessor*. The tasks of the preprocessor are to parse the SILAGE description, to perform syntax and semantic checks, to determine the datatypes of all signals and to perform a number of local transformations, common to most general purpose software compilers (as e.g. the elimination of common subexpressions). In addition to the behavioral part of the SILAGE description, the preprocessor also inputs a set of user defined allocation and assignment pragma's (which will steer the compilation process).

- *Translation/Compilation* : In order to transform the preprocessed applicative high level language description of the ALGORITHM into a customized processor STRUCTURE, the mapping tool has to assign primitive SILAGE operation to Execution Units, define the bus structure and assign the SILAGE and intermediate variables to register files and background memories. This task can be divided into a translation and a number of optimization subtasks. The translation step transforms behavioral primitives into architectural primitives. This step is of extreme importance, since it will determine how efficient the architectural properties can be exploited. In order to cope with architectural changes and expansions, this tool has to be flexible and expandable by an unexperienced user. Therefore it has been implemented in a rule based fashion. This rule base captures the knowledge of the architecture designer and addresses the real creative step in the synthesis process. The translation might be straightforwarded for a simple addition, but is far more complicated

for constructs such as multiplication (parallel, parallel-serial, constant multiplication), algorithmic delays, matrix operations, repetitions, floating point operations, double precision arithmetic, etc. A second set of rules implements the interconnection strategy. It generates the necessary busses, input multiplexers and tri-state output buffers. The current rule base for the multi-processor architecture consists of more than 100 rules, but new ones are being added regularly (as our experience grows). A user friendly knowledge acquisition system is being developed to ease the introduction of new rules.

- *Scheduling/Assignment* [Goo87] : As a result of translation step, the SILAGE description has been transformed into a data path structure and a register-transfer (RT) description of the algorithm. In the RT-description, no timing is imposed on the operations. A number of assignments (e.g. the binding of an operation to a particular EXU-instance) are also left open. The tasks of the scheduling operation thus are :

  * the ordering of the RT-operations on the time-axis in such a way that the execution of the algorithm takes a minimal number of cycles.
  * the binding of the undefined assignments so that the allocated hardware is used in an optimal way.

  A graph based scheduling tool, called ATOMICS, has been developed. Special features of ATOMICS are its capability to schedule repetitive programs and to handle I/O constraints. ATOMICS has been used to schedule extensive programs and has proven to achieve the optimal schedule in most cases.

- *Register binding/bus merging* : Once the exact timing schedule of the RT's has been determined, some extra optimization steps can be performed : the dimensioning of the register files as well as the assignment of the variables to precise register fields (based on a lifetime analysis of the variables) and the minimization of the bus count (bus merging). Procedural optimization functions have been developed for both tasks and have been integrated in the JACK-environment.

4.3 Communication Hardware Synthesis

After the derivation of the processor hardware and the controller timing, it is possible to synthesize the interprocessor communication hardware and to derive the structure and contents of the central controller. The tasks of this synthesis tool, which is currently beng developed, are to select the cheapest communication protocol (FIFO or RAM based/single or double buffering), to dimension the buffer arrays and to determine the exact timing of the control signals needed. It must be mentioned that the selection of a certain protocol can result in a number of extra constraints on the processor timing or hardware, so that a reiteration on the processor synthesis process might be needed.

5 Examples

In this section we will report on the status of the project by showing some examples. Therefore a link has been made with a module generation environment (MGE) and a floorplanning envi-

ronment (FPE). (Fig.4) The Module generation environment can be called in a procedural way to generate views of the execution units.

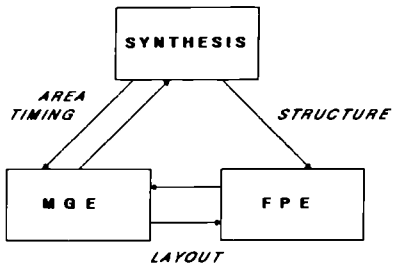


Figure 4: Link of Synthesis Tools with MGE and FPE

In an early phase of the design, the synthesis tools will ask for an area view, in order to guide the decisions made at the higher levels of the design. Also a timing view will be called. Together with the output of the scheduler (the number of clock cycles), this gives an indication of the performance of the implementation.

Once the synthesis process has reached a solution, a fully detailed layout can be produced. Therefore the complete structure is send to a floorplanning environment. From this environment, the MGE is called again to produce bounding boxes, that are used by the global router, and detailed layout can be produced very fast.

5.1 Example 1 : pcm filter

As an illustration of the CATHEDRAL-II methodology, we will synthesize the pcm filter, described by the Silage code of figure 3. In a first attempt, a single alu implementation is generated. JACK synthesizes the datapath of Fig. 5.

It consists of a ram for input and output, one alu to perform the aritmetic operations and one rom (rom ctrl) that forms the link to the controller over which immediate addresses can be fetched. (Note that the multiplications with constants have been expanded automatically in a minimal number of Add/Shift operations on the ALU.) In a first solution, the total number of machine cycles equals 33. This means that a sample rate of 300 Khz can be handled (if we assume that one cycle takes about 100ns). The bus merging algorithm is able to reduce the number of busses to 3. The generated floorplan is shown in fig. 7. The area is equal to 6.7 square mm. in a 2.5 micron CMOS process with two metal layers.

When we are looking for faster implementations, the pragma statement "pragma (*,mult,1)" can be added. This forces all multiplications to be performed on multiplier 1. After bus merging, this results in the datapath of figure 6.

This implementation runs twice as fast as the first solution (only 15 machine cycles are needed). This corresponds to an input sampling frequency of almost 670 Khz. It is interesting to note that the generation of each of these solutions (data path definition, scheduling, bus merging,

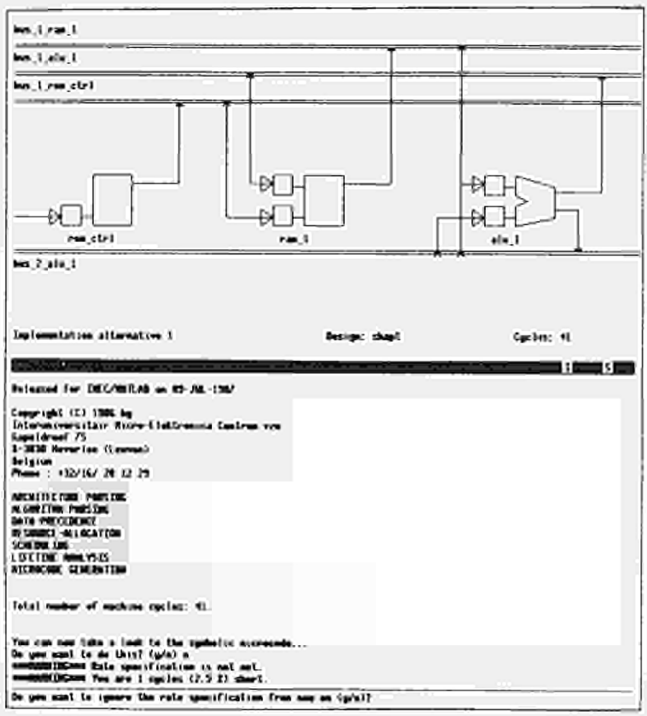


Figure 5: Synthesized datapath structure for PCM filter : single ALU based solution (including controller connection and buffer)

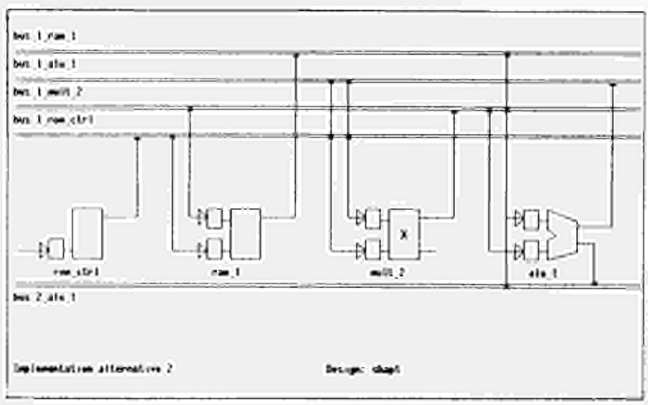


Figure 6: Synthesized datapath structure for PCM filter as in fig. 5, but with multiplier-ALU solution

area estimation) takes only a couple of minutes. This illustrates that it is possible to explore a wide design space in a short time.

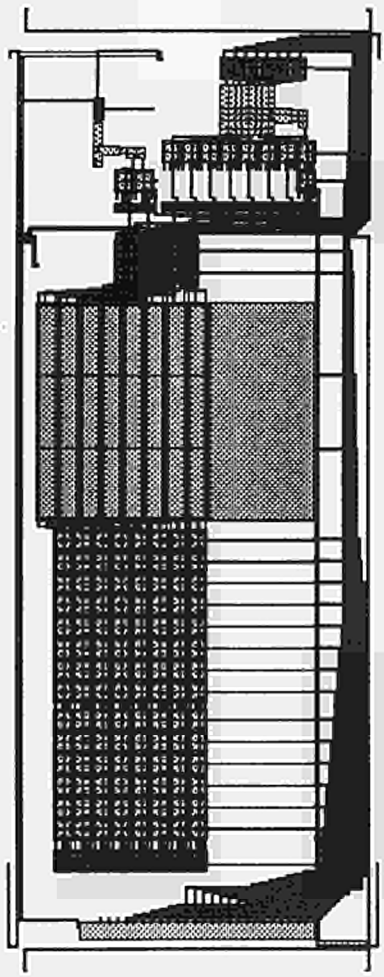


Figure 7: Generated Floorplan of PCM Filter

5.2 Example 2 : Adaptive Interpolator for Digital Audio

To demonstrate the complexity of the algorithms, which can be tackled with CATHEDRAL-II, the example of an adaptive interpolator for the correction of burst errors in digital audio [Vel83] will be discussed. The algorithm includes the computation of a 512 x 512 correlation matrix, the inversion of a 51 x 51 Toeplitz matrix using the Levinson-Durbin algorithm, the computation of the interpolation coefficients and the inversion of a full 16 x 16 matrix. Initially, a four processor

solution was proposed. A study of the complexity of the different processors and the amount of required buffer memory showed however that a one processor solution with a complex datapath was preferable (Fig. 8). The ATOMICS scheduling of the register transfer description proved that the complete algorithm can be executed in 77.000 cycles on this datapath, which easily fits within the alloted time frame of 11.6 msec.

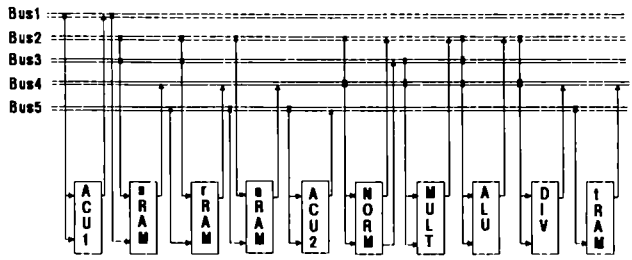


Figure 8: Datapath for Adaptive Interpolator for Digital Audio

6 Conclusion

The concepts and the methodology of an application-specific silicon compiler for complex, medium speed digital signal processing algorithms have been discussed.

CATHEDRAL-II is targeted towards a well defined customized multi-pro- cessor architecture. The operators and the mechanisms of this architecture have been captured in a translation rule base. A number of procedural optimization tasks have also been incorporated. A basic feature of CATHEDRAL-II is the openness to user interaction, where the designer can steer the synthesis process with the aid of a number of high level structural constraints. CATHEDRAL-II is being used for the generation of applications in the field of digital audio, telecommunications, speech and linear algebra (Singular Value Decomposition).

7 Acknowledgement

The authors wish to thank F. Catthoor, J. De Caluwe, G. Goossens, P. Pype and J. Vanhoof of IMEC, Heverlee, Belgium and O. Mc. Ardle of Philips Research Labs, Eindhoven, The Netherlands.

References

[Cat86] F. Catthoor et all, "General Datapath Controller and Interprocessor-communication Architectures for the Creation of a Dedicated Multi-processor Environment", IEEE ISCAS conf., May 1986, pp. 730-731.

[DeM86] H. De Man, J. Rabaey, P. Six, L. Claesen, " CATHEDRAL-II : A Silicon Compiler for Digital Signal Processing", IEEE Design and Test, Dec. 1986, pp. 13-25.

[Goo87] G. Goossens et all, "An Efficient micro-code-compiler for custom multi-processor DSP-systems", Int. ESPRIT 97 Report, 3/87.

[Hil85] P. Hilfinger, "A High Level Language and Silicon Compiler for Digital Signal Processing", Proc. IEEE CICC-conf., Portland, pp. 213-216, May 1985.

[Kow85] T. Kowalski et al, "The VLSI Design Automation Assistant : what's in a Knowledge base ?", 22nd DA Conf., pp. 252-258.

[Mar86] P. Marwedel, "A New Synthesis Algorithm for the MIMOLA Software System", 23rd design Automation Conference, pp. 271-277.

[Vel83] R. Veldhuis et al, "Adaptive Interpolation For Digital Audio Signals : An integer implementation", Int. Rep. Philips, 1983.

Project No. 97

DESIGN OF CONCURRENT SORTER NETWORKS FOR REAL-TIME IMAGE PROCESSING

U. Kleine*, R. Hofer*, K. Knauer* and I. Vandeweerd**

* Siemens AG, ZFE ME 22, Otto-Hahn-Ring 6, D-8000 Munich 83 Federal Republic of Germany

** IMEC, Kapeldreef 75, B-3030 Leuven, Belgium

In this contribution the design of dedicated concurrent sorter networks for real-time robot vision and bio-medical applications is described. Due to the high sampling rates (5 MHz - 40 MHz) and the large data streams, the implementation of image processing algorithms demands for dedicated chips. In order to reduce the design costs for an individual application, one of the objectives within the ESPRIT 97 project is to develop a set of flexible parameterizable modules which makes the implementation of different image processing algorithms possible. This design strategy will be illustrated with a sorter module for rank order filtering. A bit-parallel 'odd-even transposition sort' and a bit-serial 'odd-even merge' algorithm have been selected for an implementation. A module generator environment which has been developed within this project was used to create various sorter networks with different number of input words. A 16 and a 25 input word sorter network using Batcher's 'odd-even merge' algorithm and a 25 input word 'odd-even-transposition' sorter network will be described.

1. INTRODUCTION

Although the high cost of today's image processing systems limits the number of possible applications substantially, the field of digital image processing is rapidly growing. It is one of the objectives within the ESPRIT 97 project to find architectures for certain special image processing operations, that can be implemented on one or perhaps several chips so that a low cost digital image processing system can be realized. In order to reduce design time and design cost, a set of flexible modules has been defined. These modules can be generated automatically using the module generator environment which has also been developed within this project [1,2].

In picture processing, especially at the front end of a system, many operations are performed repeatedly over a large number of

pixels of an image. Thus in most cases parallel processing can be applied. This can be done with systolic arrays with a regular and a modular organisation. With systolic arrays high computation throughput can be achieved by pipelining and multiprocessing. Since only nearest neighbourhood communication is allowed, the throughput rate is not degraded. The data stream and the control flow of the arrays should be kept as simple as possible. However, for many digital signal processing algorithms, especially for efficient sorting algorithms, the requirement of nearest neighbourhood communication only is too restrictive. Therefore a more general type of arrays has been used called local systolic arrays. For local systolic arrays also non-neighbourhood communication is allowed, if in contrast to a standard cell design the wiring is predefined by the algorithm to be implemented.

In the following, the proposed design style will be demonstrated with an example of a sorter module for rank order filtering. Rank order filtering is a nonlinear signal processing technique used in many image processing systems. Rank order filters are used for noise reduction and image enhancement, for example in robot vision systems.
They were first suggested by Tukey [3] for pitch extraction in speech analysis and were later adapted for image processing. The most popular rank order filters are the median filters [4]. Also combinations of linear and rank order filters have been reported [5-7]. Rank order filters consists of a sliding window encompassing a number of pixels. The pixel in the center of the window is replaced by the pixel of rank k. In each clock cycle the pixel of rank k is determined by the pixels within the window. For instance the median pixel has the rank $k = (N+1)/2$, where N is the odd number of pixels within the window. Thus, rank order filters consists basically of a sorter. In the next paragraph some properties of two-dimensional rank order filters are described in more detail.
In Paragraph 3 a rank order filter architecture is presented which includes its own memory. This is a real-time stand-alone processing system needing no additional hardware. Sometimes, even for a certain application environment, different window sizes and forms can be required. Therefore an architecture with a programmable window is used. Several sorting algorithms have been inspected with respect to a hardware implementation. The selected sorting algorithms are based on Batcher's 'odd even merge' algorithm and on the 'odd even transposition' algorithm. The chip area of the first is proportional to $N(\log_2 N)^2$ while the second is proportional to N^2. These sorters will be described in more detail in Paragraph 4. A module generator environment [1,2] and symbolic layout techniques [8] have been applied. The creation of different sorters using these techniques will be described in Paragraph 5.

2. PROPERTIES OF TWO-DIMENSIONAL RANK ORDER FILTERS

In the last years, much effort has been invested in the analysis of rank order filters [4-7,9-16]. Because the median filter is the most important rank order filter, its properties are briefly described. Due to their nonlinear nature median filters cannot

be characterized in the frequency domain. The only way to quantify their performance is to use different noise sources like uncorrelated white noise, Gaussian noise or impulsive noise. As an example an average filter with a n x n window size reduces the variance of white noise by a factor n^2, regardless of the actual noise distribution. Median filters, on the other hand, yield different variances depending on the parent noise distribution. For a median filter with a window size of n x n, an approximate relation for the variance σ of the median of n^2 independent points from a density function $p_x(x)$ is [17]

$$\sigma^2(n \times n) = \frac{1}{4(n^2+2)} * \frac{1}{[p_x(x)]^2} \quad . \qquad (1)$$

The variance strongly depends on the original noise distribution $p_x(x)$, as can be seen from Eq. 1. In the following, the filtering capability of median filters will be illustrated by some examples.

Fig. 1a shows an original image, which consists of 512 * 512 pixels with a resolution of eight bits. In Fig. 1b Gaussian noise has been added ($N(\mu = 0, \sigma = 67)$ and in Fig. 1c impulsive noise (salt and pepper noise) has been added using the model of [18] with p = 0.3 for an interval of [0,255]. On these noisy pictures median filtering has been applied using rectangular, xshape and cross-shape windows of size 5 x 5 (see Figs. 2 and 3). As can be seen, the filtering capability of median filters depends very much on the shape of the noise distribution $p_x(x)$. Therefore the filtering effect for impulsive noise is good while for white or Gaussian noise it is rather poor.
Another property of median filters is that if an image is repeatedly filtered, it will converge to an image invariant to further filtering, called root signal. The statistics of the roots are taken as a performance criterion for the system specification.
To summarize the performance, median filters behave as low-pass filters and at the same time preserve important image structures, such as edges.
Some closely related filter structures have also been reported in literature [4-6]. For instance combinations of linear and median filters, weighted-median filters in which the center pixel is given more weight.
Another median filter type is the recursive filter in which the median pixels are used for further filtering. In general, rank order filters could also be used for shrinking or expanding objects in a picture. Consider the gradient picture of Fig. 1a in Fig. 4a. Fig. 4b illustrates rank order filtering with a 5 x 5 window and taking the pixel with rank 5 as output. As can be seen the gradients are expanded. Edge detection can also be performed with rank order filters by simply taking the difference of pixels with minimum and maximum rank. Another interesting application is edge gradient enhancement, described in [7]. In the next Paragraph the general chip architecture of a rank order filter will be described.

Fig. 1a: Original image

Fig. 1b: Original image with Gaussian noise (N(0,63))

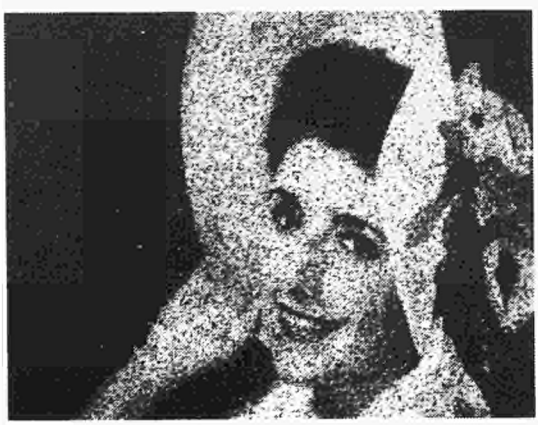


Fig. 1c: Original image with impulsive noise [18] (p=0.3 and intervall of [0, 255]

Fig. 2a: Result of median filtering of Fig. 1b with a square window of size 5x5

Fig. 2b: Result of median filtering of Fig. 1b with a xshape window of size 5x5

Fig. 2c: Result of median filtering of Fig. 1b with a cross-shape window of size 5x5

Fig. 3a: Result of median filtering of Fig. 1c with a square window of size 5x5

Fig. 3b: Result of median filtering of Fig. 1c with a xshape window of size 5x5

Fig. 3c: Result of median filtering of Fig. 1c with a cross-shape window of size 5x5

Fig. 4a: Gradient image of Fig. 1a

Fig. 4b: Result of rank order filtering with a square window of size 5x5 and rank = 5

3. ARCHITECTURE AND DATA FORMAT OF A RANK ORDER FILTER

A modular target architecture has been chosen in order to permit a flexible design. It consists of 4 modules, a line buffer, a RAM, a sorter and a clock and control unit. Each of these blocks will be optimized separately. Since the sorter is the core piece of the chip, it is described in more detail.
The data are supplied in a raster scan format (line by line scanning). Each sample or pixel is coded in a digital 8 bit word. The line length can vary from 512 to 1024 pixels, the system clock is assumed to be between 5 MHz and 16 MHz and the window size can range to 7 x 7 pixels.
Fig. 5 shows the block diagram of the rank order filter. In the following the different modules will be described.
The line buffer is a memory in which n-1 adjacent lines of the picture can be stored. In the case of a 7 x 7 window the line buffer size is 24576 or 49152 bits for 512 or 1024 pixels per

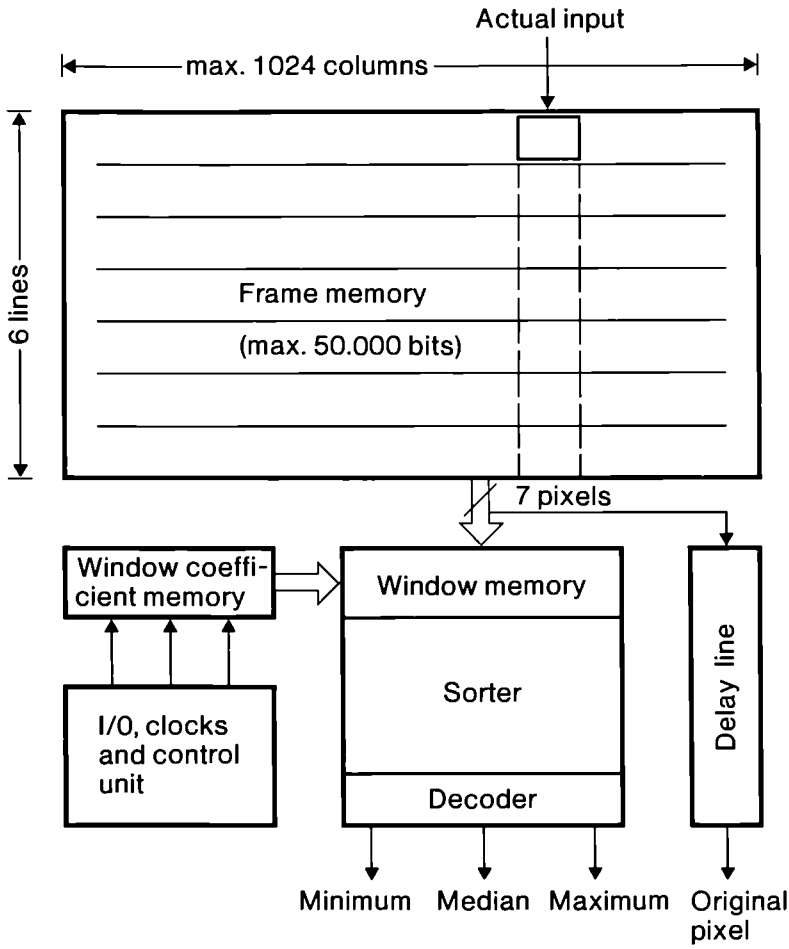


Fig. 5: Block diagram of the rank order filter chip

line, respectively. The n-1 output pixels of the line buffer together with the updating pixel, build an n pixel column of the input picture. This column updates the window memory each cycle. The buffer could be built with an 8 bit shift register [19] or with a dynamic memory and a pointer [20,21]. The last solution uses a 3-transistor cell and is preferable to the first one considering power consumption and chip area.

The RAM is used as an instruction register. It contains the actual window form and the ranks of the two selected outputs. Various window forms can be used. Such forms are horizontal, vertical or diagonal line segments, squares and square rings, crosses, rectangles, circles and circular rings (see Fig. 6). Due to the definition of median filters, the actual window form must be symmetrical around the center pixel. Therefore, only 25 bits are required for a 7 x 7 window. Half of the pixels, which lie outside the actual window, are set to the maximum value and

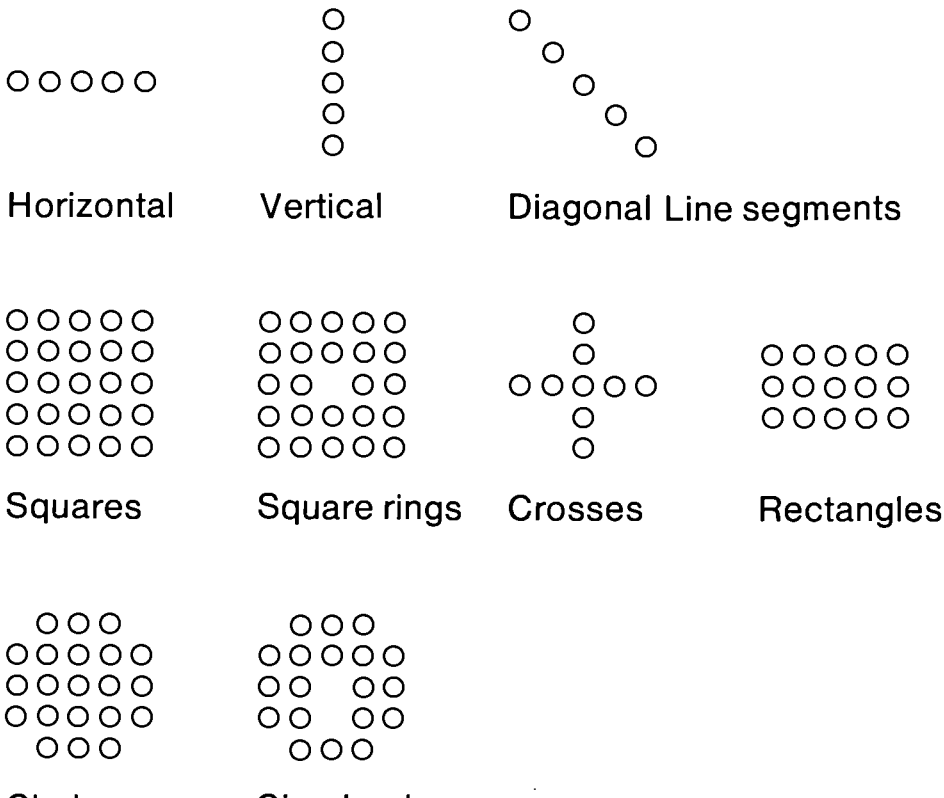


Fig. 6: Common window forms

the other half outside the actual window, are set to the minimum value.
The clock and control unit provides the necessary clock signals and control data for the rank order filter chip. To synchronize the clock signals a PLL-based clock generator [22] has been selected for the bit-serial sorter.
The sorter module consists of four parts: the window memory, the synchronization line, the decoder and the sorting network.
The window memory strongly depends on the sorter network. Its function is to build the window from the updating column and to provide the output of the 7 x 7 pixels of the window. In other words, it has to adapt the frame memory to the sorting network. It consists of a number of resettable shift registers. These resettable registers are controlled by the window form bits and additional control bits. The control bits are used for calculating the border pixels.
The synchronization line consists of an 8 bit shift register with a length equal to the delay of the sorting network. It provides the input image synchronous with the filtered image to perform further calculation.
With the rank decoder block several outputs with arbitrary rank can be selected.
The sorter must be able to compute one median each cycle. Thus only very regular algorithms can be taken into consideration. Two different sorting networks are described in the next paragraph.

4. SORTER ARCHITECTURES

Effective sorting algorithms have been studied for a longer time [23]. Because most of these algorithms have been developed for a general purpose computer, only few of them are suitable for a word-parallel VLSI implementation. Suitable for a VLSI implementation means that the data flow of these algorithms should be regular and the control of this flow should not be too complicated.

In the following, a lower bound on the number of operations will be derived. Assume that the basic operation of a sorting algorithm is a comparison of two numbers followed by a conditional exchange, it can be shown that a lower bound for the maximum number of steps in a sort of N numbers is ($\log_2$ N!) [23]. Using Stirling's approximation for N! [23], the lower bound for the maximum number of operations grows as

$$n_c \approx N \log_2 N, \qquad (2)$$

where n_c is the number of comparisons. If N numbers or pixels have to be sorted, up to N/2 comparisons can be done simultaneously assuming a two input compare and exchange (swap) operation. Fig. 7 shows the block diagram of a compare and swap

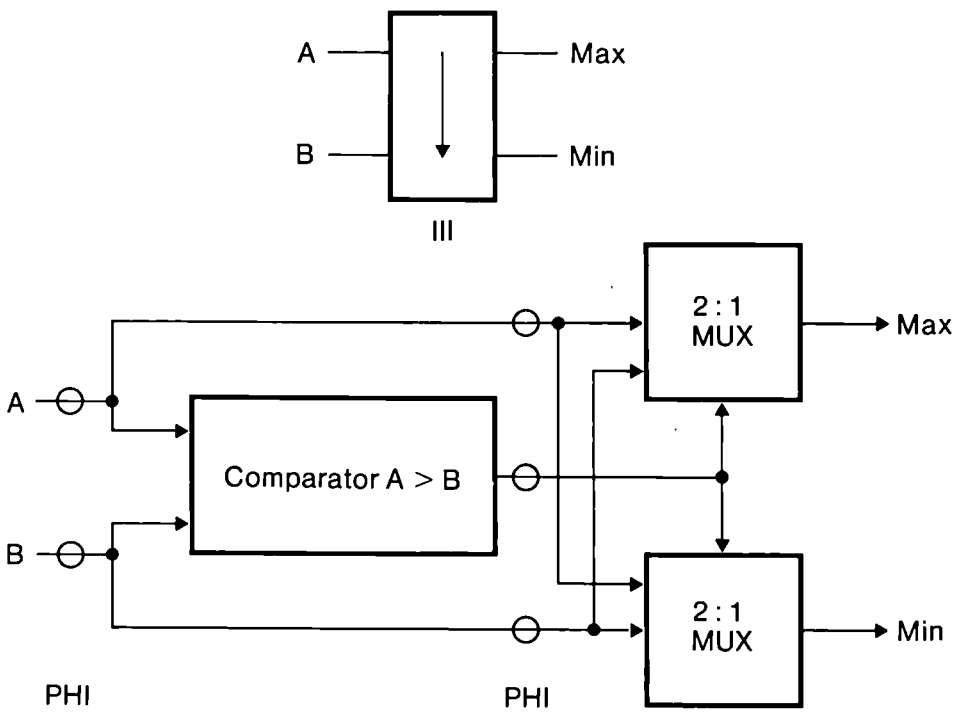


Fig. 7: Block diagram of a compare and swap unit

unit. This unit is the basic processor element for the implementation of sorting networks. It consists of a comparator followed by two 2:1 multiplexers, which pass the larger of the two inputs to output MAX and the smaller to output MIN.

Because different window forms are allowed, e. g. crosses, no presorting of the incoming pixels has been assumed. The pixels of the actual window are first reorganized in a vector and then sorted in parallel. In the following two different sorting algorithms are described.

4.1 ODD-EVEN-TRANSPOSITION SORT

A simple sorting algorithm is the well-known bubble-sort algorithm [23], also known as 'odd even transposition' sort (if the sorting is carried out in parallel). The implementation of this algorithm only requires a compare and swap unit and a simple delay unit with the same latency time as the compare and swap unit. Fig. 8 shows a floorplan of the 'odd even transposition' sorter for a rectangular window of 3 x 3. The pixels are simultaneously loaded into the front end of the sorting network. The sorting is carried out by successive permutations. Each row of the sorting network reduces the number of possible initial permutations i! to (i-1)!. Thus N stages are required to sort N pixels. The number of basic compare and swap operations is then determined by

$$n_{cs} = \lceil (N-1)/2 \rceil * N, \qquad (3)$$

where $\lceil x \rceil$ stands for integer of x.
The sorter of Fig. 8 is fully pipelined. One median value is computed every clock cycle. The cells can be connected by abutment and the required chip area of the odd-even-transposition sorter is proportional to N^2.

4.2 ODD-EVEN MERGER ARCHITECTURE

The next parallel sorting network to be described is based on Batcher's 'odd-even merge' algorithm [24,25]. The sorting network consists of a cascade of mergers. A merger is an element that produces one sorted sequence from two sorted input sequences of pixels. Starting with N/2 two-input mergers, the sorted results are fed into N/4 four-input mergers, and so on until finally all pixels are sorted. The individual mergers can be constructed inductively.
Assume two sorted sequences $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_m \rangle$, where n is a power of 2 and $m \leq n$. The odd elements and the even elements of the two sequences are sorted separately, resulting in two new sorted sequences $\langle c_1, \ldots, c_{m/2+n/2} \rangle$ and $\langle d_1, \ldots, d_{m/2+n/2} \rangle$. This step is performed recursively. Then, applying 'compare and swap' operations on the sequences C and D, the output pixels will be sorted. The algorithm delivers best results if the number of pixels to be sorted is a power of 2. But the algorithm can also be adapted for an arbitrary number of inputs [23,25]. Fig. 9 shows a floorplan of such a sorter with N = 25. As can be seen a very regular network emerges. The wiring between two consecutive comparator stages could be routed either

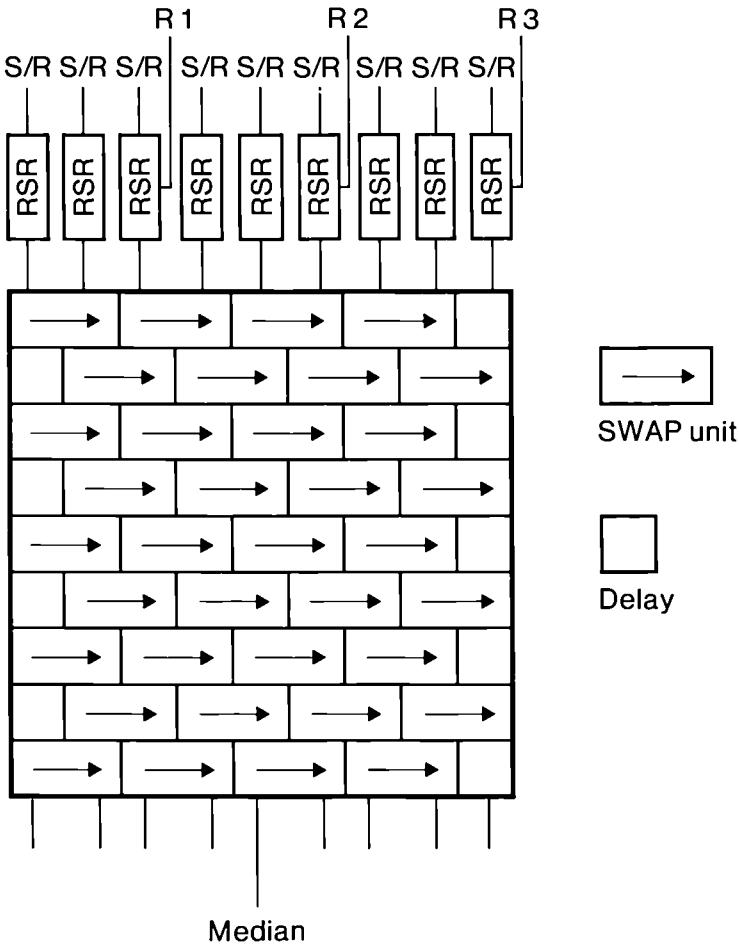


Fig. 8: Floorplan of a 'odd even transposition' sorter for a 3x3 window

with a perfect shuffle [26] or with two layer river routing. The number of the required sorting stages for the odd-even merger is

$$n_m = (\lceil \log_2 N \rceil)\,(\lceil \log_2 N \rceil + 1)/2, \qquad (4)$$

thus the required chip area is proportional to N(log2N)[(log2N)+1] if the area for routing is ignored.

5. CIRCUIT DESIGN

The basic computation unit is the compare and swap unit. If bit-serial computation is possible with a given technology, it is

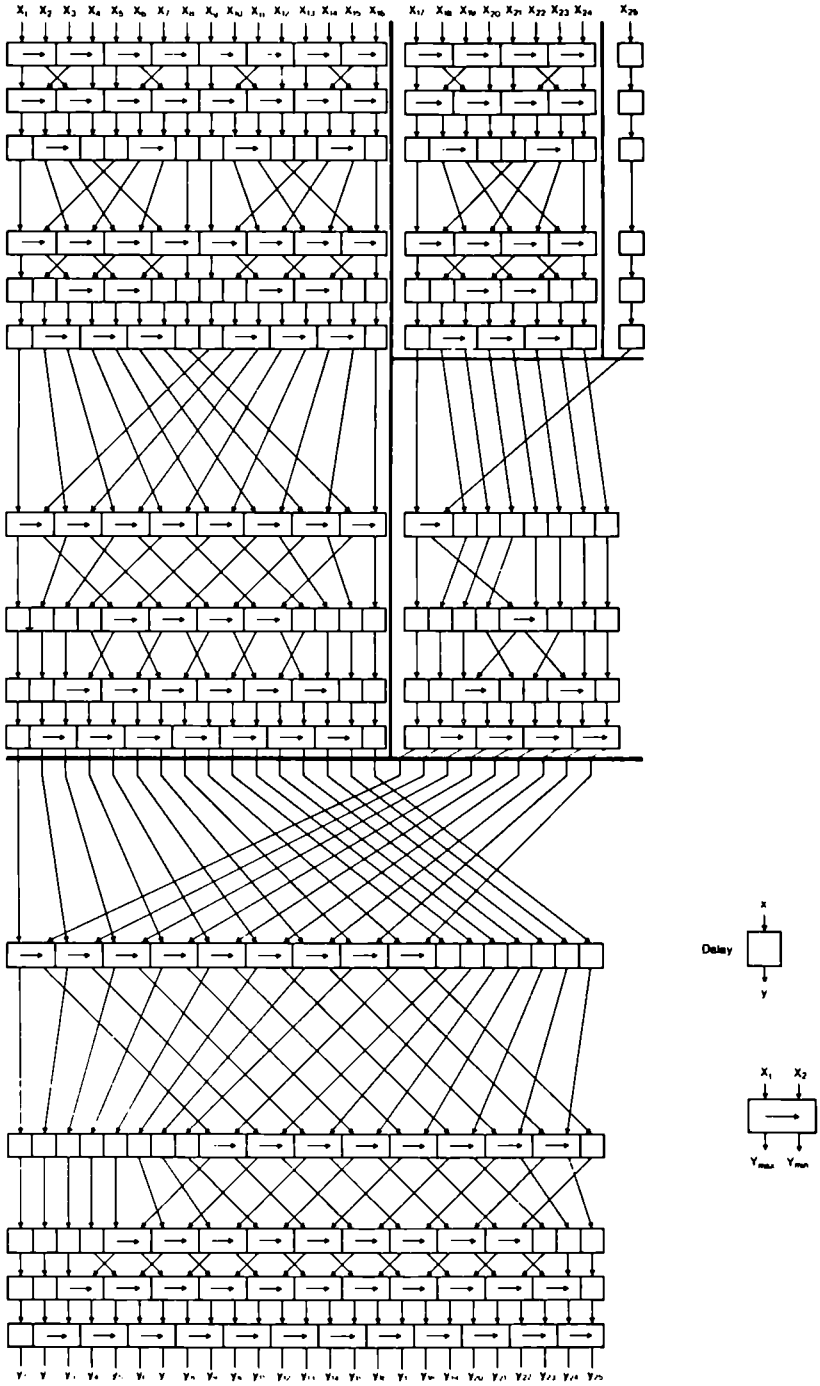


Fig. 9: Floorplan of a 'odd even merge' sorter with 25 inputs

advantagous with respect to the chip area to perform the computation bit-serially. The required area per bit is about two times larger in a bit serial approach than in a bit-parallel one. Thus the total area of the bit-serial sorting network will be about four times smaller than the bit-parallel area assuming 8 bit wordlength. On the other hand the operation speed of the bit-serial computation unit is 8 times higher and the estimated power consumption will be about two times larger. To test the performance of bit-serial and bit-parallel sorters a bit-parallel 'odd-even-transposition' sorter and a bit serial 'odd-even merge' sorter have been selected for an implementation. As stated in the introduction a module generator environment (MGE) has been used in order to reduce the design time.

5.1 MODULE GENERATION

Large parts of the design have been automatically synthesized. In the case of the sorter design, module generation is especially suited, because of the high complexity of the synthesis algorithm. This in contrast to most modules (e. g. ALU's, PLA's, ROM's.....) that can be generated by fairly simple procedures. Creation of a module generator requires additional effort compared to the design of an instance. This overhead, resulting in extra costs, should be minimized and spread out over different designs as much as possible.

To minimize the generator creation effort, MGE has been used, offering some interesting features.

* It allows for a recursive definition of the generator. This makes it possible to map the sorting algorithm directly into the synthesis algorithm. As a result, the creation and verification efforts are reduced significantly.

* It hides details such as cell size and exact coordinates at which they should be placed by making use of compaction and routing algorithms. Not only a reduction in creation effort is achieved, but it also offers an increased flexibility.

In order to spread out the creation cost of the module generator over several design instances, it is desirable to obtain an increased generator lifetime and a high degree of reusability.

* Increased lifetime requires flexibility to survive technology updates. By making use of CAMELEON (sticks editor and compaction [8]) to perform the leaf cell design, nondrastic changes in technology can be handled automatically. Severe technology changes (e. g. PWELL to NWELL) might require manual intervention at the stick diagrams level.

* A high degree of reusability can be accomplished by allowing for a wide range of parameter values. In case of the sorter the parameter is the number of words to be sorted. This value can range from 2 up to 49. The range incorporates most of the practical values for our applications.

Fig. 10 shows an example of an odd-even merge sorter with 16 inputs before compaction and routing. The input procedure of the sorter is a compact recursive definition written in LISP which

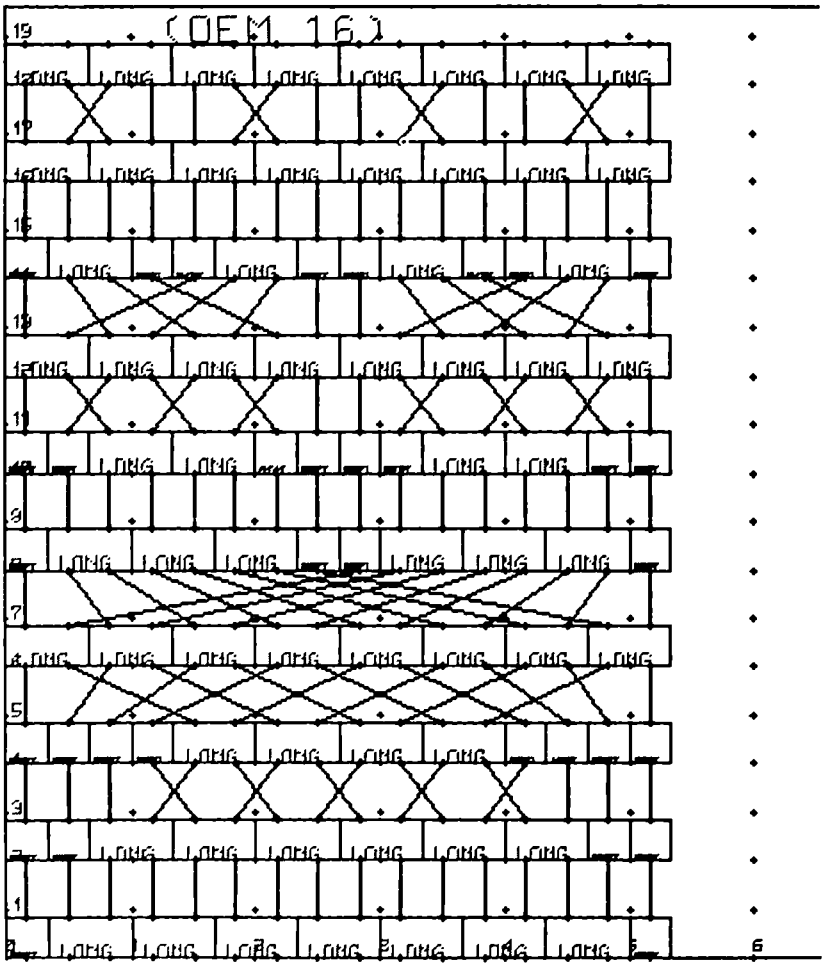


Fig. 10: Example of a 16 input sorter

describes the relative placement of the cells and their connections. The routing and the compaction is then performed automatically.

5.2 LEAF CELL DESIGN

One way to build a bit-parallel comparator is to use an adder and a complementer. But since an adder circuit is more complex than a comparator circuit, a symmetrical CMOS comparator has been used for the bit-parallel odd-even-transposition sorter. Fig. 11 shows the circuit diagram of this comparator leaf cell. A bit-parallel comparator for the words A and B can be assembled

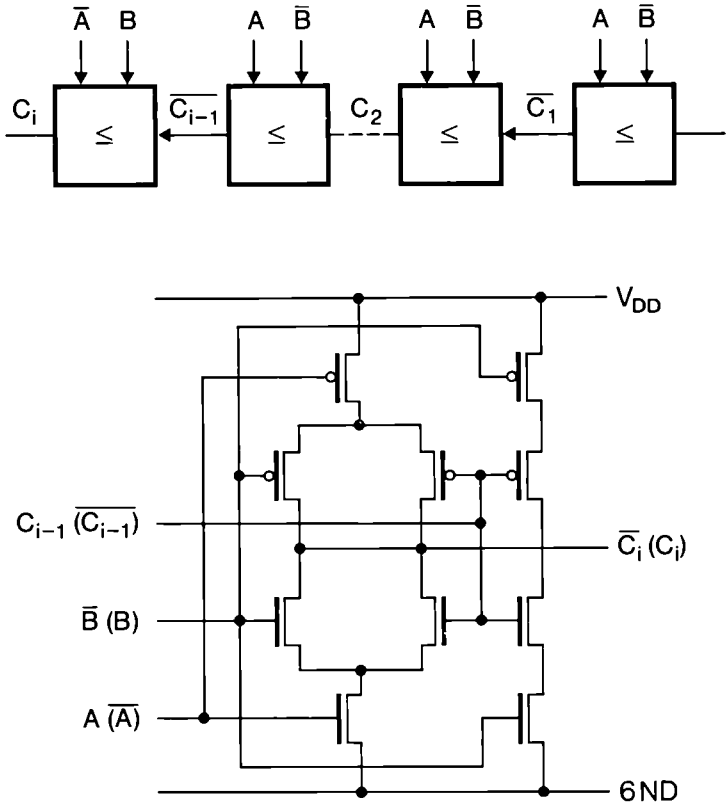


Fig. 11: Circuit diagram of a bit-parallel comparator

using this circuit. Since the 'greater or equal' signal in this comparator will be alternated, inverted and noninverted, the single bits of the words A and B have to be processed in a similar way.

The full compare and swap unit has been designed using CAMELEON [8]. The area is 280 x 240 µm² for a 1.5 µm CMOS PWELL process. In the bit-parallel compare and swap units, the swapping of the inputs can only be done after the result of the entire 8 bit comparison has been computed. Note that a bit-parallel comparator starts the calculation with the LSB's. This requires a signal propagation through eight stages of the bitwise comparator. However, it is possible to eliminate this propagation delay by using a bit-serial architecture. The bit-serial comparator starts the computation with the MSB's. This invokes breaking up the compare and swap operation into 8 steps. Fig. 12 shows a serial compare and swap unit with two levels of pipelining, which allows operation cycles of 5 nsec in a 2 µm CMOS technology. The comparator is built with a finite-state machine in which the greater G_i and the smaller L_i states are stored. For the comparator the following Boolean equations can be given:

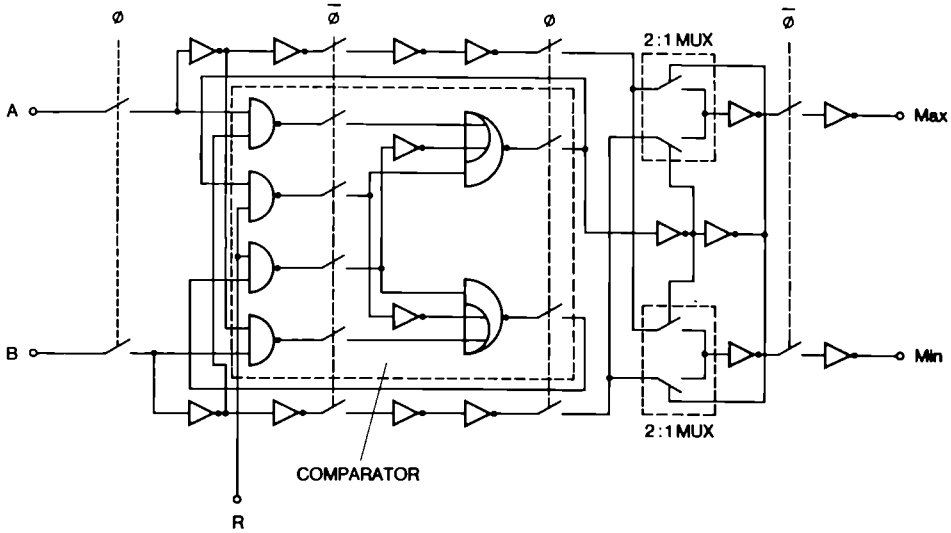


Fig. 12: Logic diagram of a bit-serial compare and swap unit

$$G_i = \overline{[(\overline{\overline{A} * \overline{B}}) + (L_{i-1} * R)] * [\overline{G_{i-1} * R}]} \qquad (5\ a)$$

$$L_i = \overline{[(\overline{\overline{A} * B}) + (G_{i-1} * R)] * [\overline{L_{i-1} * R}]} \qquad (5\ b)$$

for i = 1, ..., 8. R is the reset signal, with R = 0 for i = 1 and R = 1 otherwise. This cell has been layed out in a 2 µm CMOS process and the area is 178 x 236 µm. The cell is currently being redesigned using single clock registers as shown in Fig. 13 in a 1.5 µm CMOS technology.

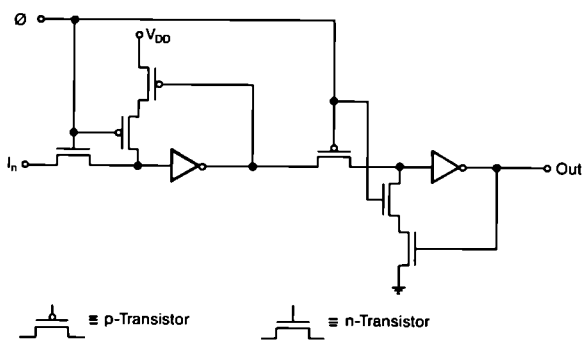


Fig. 13: Circuit diagram of a single clock register

5.3 TESTING

A self test strategy has been developed to verify the functionality of the sorter networks. No additional test aids are included inside the sorter networks in order not to lower the performance and not to increase the required area. Therefore the sorter networks are tested as a whole.

Although there exist N! possible initial condition, only 2^N test patterns are required when the 0 - 1 principle [23] is applied. The 0 - 1 principle says that for a circuit consisting solely of comparators, it suffices to verify its correct operation if the value of each input is restricted to be either zero or one. But actually [27] for the bit-serial 25 input word sorter only 225 deterministing test patterns must be applied to detect all stuck-at faults and 72,8 % of the stuck-open faults. To simplify the generation of the test patterns, a pseudo-random test pattern approach was also considered. The results are listed in Tab. I [27]. To generate the pseudo-random pattern two independent feed back shift registers are added at the front end of the sorter networks and a signature register at the output of the sorting networks for the evaluation. This test strategy requires a minimum hardware overhead. For the redesign of the bit-serial sorter in a 1.5 CMOS technology a special layout strategy will be used [27] which may either avoid or make stuck open faults easier detectable. The testability results are also included in Tab. I (third column).

Sorter network with 25 inputs	stuck-at	stuck-open (original)	stuck-open (redesign)
# faults	5918	4865	3892 (80%)
4520 pseudo FC ramdom pattern #udf	100% 0	83.6% 801	99.6% 17
225 determ. FC pattern #udf	100% 0	72.8% 1323	94.3% 223

#udf: number of undetected faults

Tab. I: Simulation results of the bit-serial sorter with 25 inputs [27]

5.4 LAYOUT

Several sorters have been automatically generated. Fig. 14 shows the layout of the bit-parallel 'odd even transposition' sorter for 25 inputs. As can be seen a very regular layout occurs. The stick-diagram approach which eases the topologic and global optimizations yields a fairly good density of about 4500 transistors/mm². The whole sorter occupies an area of 3.45 * 6.1 mm² in a 1.5 µm CMOS technology. Two layout examples of the bit-serial 'odd even merge' sorter including the reset line for 16 and

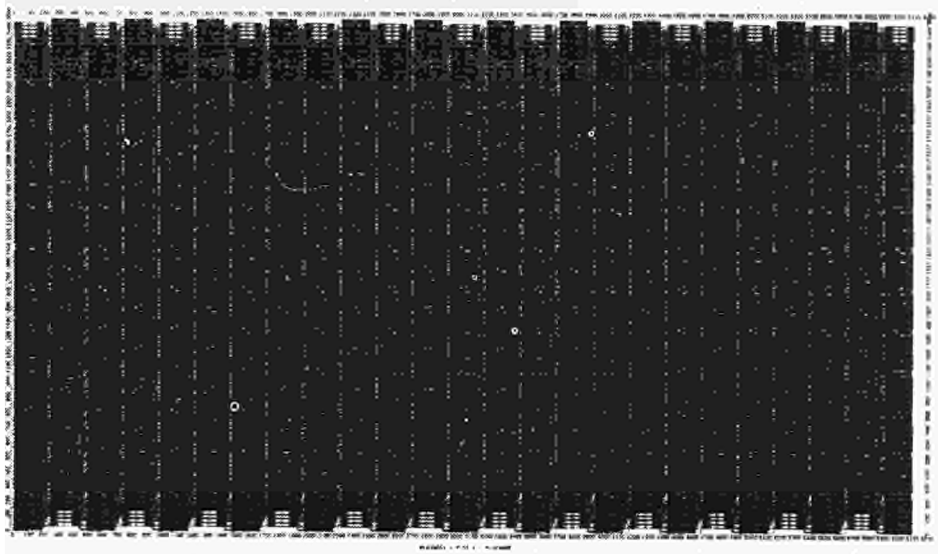


Fig. 14: Layout of the bit-parallel 'odd even transposition' sorter with 25 inputs

25 inputs are shown in Fig. 15. The required areas are 2.45 * 1.85 mm² and 4.25 * 2.9 mm² respectively in a 2 µm CMOS technology. The transistor density is higher than 2000 transistors/mm² and as can be seen from the Figs. 15 the routing area can be neglected. The whole rank order filter of Fig. 5 is currently being designed. The chip will contain about 200 000 transistors.

6. CONCLUSIONS

The design of complex, high performance dedicated chips for real-time image processing using flexible modules has been described. As an example of the proposed methodology the design of parameterizable modules for two-dimensional rank order filters using a module generator environment and symbolic layout techniques has been demonstrated. Parallel processing and pipelining have been applied. Very regular architectures for a bit-parallel 'odd even transposition' sorter and a bit-serial 'odd even merge' sorter have been presented. A self test strategy has been developed for the modules. With the presented design methodology, dense layouts are achieved in relatively short design times.

ACKNOWLEDGEMENTS

This work was supported by the EC ESPRIT'97 project. The authors wishes to thank E. De Man, S. Köppe, and T. Noll for valuable discussions.

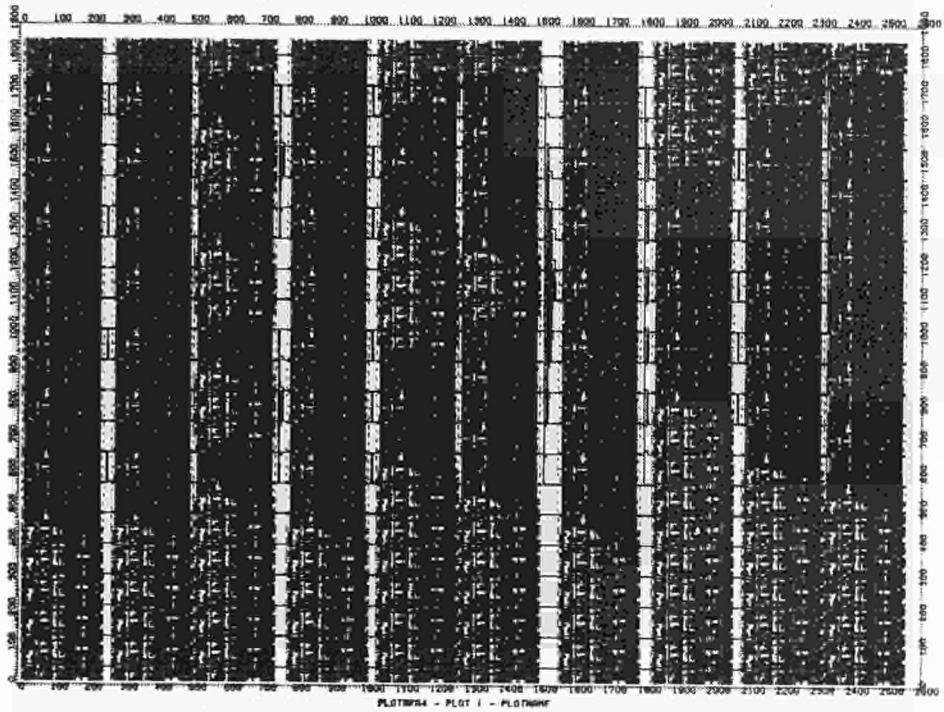


Fig. 15a: Layout of the bit-serial 'odd even merge' sorter with 16 inputs

REFERENCES

[1] P. Six, L. Claesen, J. Rabaey, and H. De Man, "An intelligent Module Generator Environment", DAC `86, pp. 730 - 735, 1986.

[2] I. Vandeweerd, "Module Generator Environment Reference Manual", ESPRIT `97, interim report, IMEC, Feb. 1986.

[3] J. W. Tukey, "Exploratory Data Analysis", Addison-Wesley, Reading, Mass., 1977, preliminary ed. 1971.

[4] T. S. Huang, ed., "Two-Dimensional Digital Signal Processing II", Springer-Verlag, Berlin, Heidelberg, New York, 1981.

[5] Y. H. Lee and S. A. Kassam, "Generalized Median Filtering and Related Nonlinear Filtering Techniques", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-33, No. 3, pp. 672 - 683, June 1985.

[6] I. Pitas and A. N. Venetsanopoulos, "Nonlinear Order Statistic Filters for Image Filtering and Edge Detection", Signal Processing, No. 10, pp. 395 - 416, 1986.

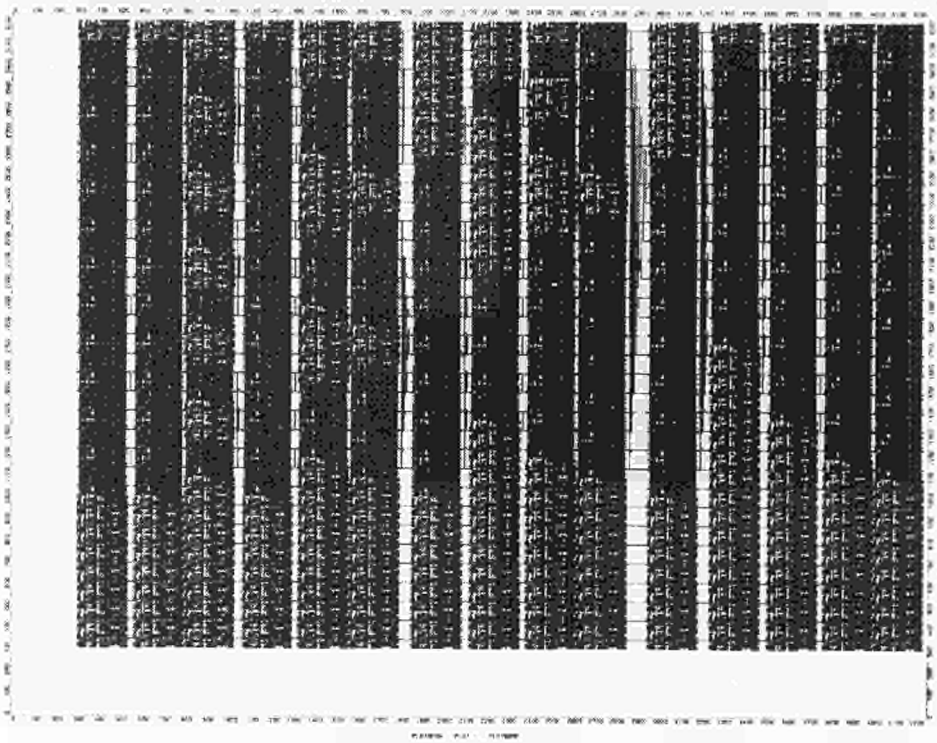


Fig. 15b: Layout of the bit-serial 'odd even merge' sorter with 25 inputs

[7] Y. H. Lee and A. T. Fam, "An Edge Gradient Enhancing Adaptive Order Statistic Filter", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-35, No. 5, pp. 680 - 695, May 1987.

[8] K. Croes, H. De Man, and P. Six, "CAMELEON, a Process Tolerant Symbolic Layout System", ESSCIRC `87, Bad Soden, 1987.

[9] N. C. Gallangher and G. L. Wise, " A Theoretical Analysis of the Properties of Median Filters", IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-29, No. 6, pp. 1136 - 1141, Dec. 1981.

[10] J.P. Fitch, E.J. Coyle, and N.C. Gallangher, "Root Poperties and Convergence Rates of Median Filters," IEEE Trans. Acoustic, Speech, and Signal Processing, Vol. ASSP-33, No. 1, pp. 230-240, Feb. 1985.

[11] P. M. Narendra, "A Separable Median Filter for Image Noise Smoothing," Proc. 1978 Conf. Pattern Recognition and Image Processing, Chicago, IL, pp. 137-141, May 1978.

[12] G.R. Arce, and R.J. Crinon, "Median Filters: Analysis for 2-Dimensional Recursively Filtered Signals," Proc. IEEE Int. Conf. on Acoustic,Speech, and Signal Processing, San Diego, pp. 23.11.1-23.11.4, March 1984.

[13] T.A. Nodes, G.Y. Liao, and N.C. Gallangher, Jr., "Statistical Analysis of Two-Dimensional Median Filtered Images," Proc. IEEE Int. Conf. on Acoustic, Speech, and Signal Processing, San Diego, pp. 23.2.1-23.2.4, March 1984.

[14] R.L. Kirlin, B. Cudzilo, and S. Wilson, "Two-Dimensional Orthogonal Median Filters and Applications," Proc IEEE Int. Conf. on Acoustic, Speech, and Signal Processing, Tampa, FLD, pp. 1325-1328, March 1985.

[15] A.R. Butz, "A Class of Rank Order Smoothers," IEEE Trans. Acoustic, Speech, and Signal Processing, Vol. ASSP-33, No. 3, pp. 672-683, June 1985.

[16] M.I. Shamos, "Robust Picture Processing Operators and their Implementation as Circuits," Proc. Image Understanding Workshop, Pittsburgh, PA, pp. 127-129, Nov. 1978.

[17] P.M. Narendra, "A Separable Median Filter for Image Noise Smoothing," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 1, pp. 20-29, Jan. 1981.

[18] B.I. Justusson, "Median Filtering: Statistical Properties," in T.S. Huang, Ed., "Two-Dimensional Digital Signal Processing II," Springer Verlag, Berlin, Heidelberg, New York, pp. 169-173, 1981.

[19] N. Demassieux, E. Jutland, M. Saint-Paul, and M. Dana, "VLSI Architecture for a ONE-Chip Video Median Filter," Proc. ICASSP'85, Tampa, FLD, pp. 1001-1004, March 1984.

[20] B. Zehner, H.J. Mattausch, R. Tielert, and H.J. Grallert, "A Two-Dimensional Digital Filter for TV-Pictures," ISSCC'86, Anaheim, CA, pp. 146-147, 330, Feb. 1986.

[21] P.A. Ruetz and R.W. Brodersen, "Architectures and Design Techniques for Real-Time Image-Processing IC's," IEEE J. of Solid-State Circuits, Vol. SC-22, No. 2, pp. 233-250, April 1987.

[22] D.-K. Jeong, G. Borriello, D.A. Hodges, and R.H. Katz, "Design of PLL-Based Clock Generation Circuits," IEEE Solid-State Circuits, Vol. SC-22, No. 2, pp. 255-261, April 1987.

[23] D.E. Knuth, "The Art of Computer Programming," Vol. 3, "Sorting and Searching", Reading, MA, Addison-Wesley, 1983.

[24] K.E. Batcher, "Sorting Networks and Their Applications," Proc. AFIPS 1968 SJCC, Vol. 32 CAFIPS Press, Montnale, NJ, pp. 307-314, 1968.

[25] U. Kleine, "Novel Sorter Architecture for Image Processing Rank Order Filters," Electronic Letters, Vol. 23, No. 1, pp. 45-46, Jan. 1987.

[26] H.S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Computers, Vol. C-20, pp. 151-161, 1971.

[27] S. Koeppe, "Optimal Layout to Avoid CMOS Stuck-Open Faults," DAC'87, Miami Beach, FL, pp. 829-835, June 1987.

Project No. 97

ALCATEL-BTM LAYOUT AND FLOORPLAN METHODOLOGY

L. Cloetens, J. Goubert and P.P. Guebels Alcatel-BTM.
Micro-electronics Development Group
Francis Wellesplein 1, Antwerp, Belgium

True Silicon Compilation is a significant achievement of the Esprit 97 project. One of the key concepts which has successfully driven the program is the structuring of the synthesis task in two parts. Architectural synthesis from an algorithmic description and automatic generation of mask geometry. Both synthesis activities communicate through an interface, defined as the meet in the middle plug. (fig.1) The purpose of the paper is to report the Alcatel-BTM implementation of a layout and floorplan compilation into silicon which is a direct result of its Esprit 97 partnership.

1. INTRODUCTION

Most existing floorplanning methodologies do not allow to take into account the effect of the toplevel routing on the floorplan, which very often results in considerable area increase after realisation of the sub-locks. This increase is not only caused by the routing area but especially by the fact that the floorplan becomes inoptimal. Redesign of the sub-locks in this stage of the design to fit better in the floorplan requires a major effort.
The Alcatel-BTM floorplanning methodology starts with an estimate of the block sizes and terminal positions of the different sub-blocks. This information is supplied by the designer or the higher level tools. With this information an optimal floorplan is obtained by trying out different placements using a place and route tool. The size of the blocks and the terminal positions are optimized in function of the floorplan. Finally the floorplan is gradually refined as the exact layout of the different sub-blocks becomes available. This results in a design with the toplevel routing and the placement completed together with the layout of the sub-blocks.

Layout of the functional building blocks can be provided both by the designer (full custom) or by blockgenerators (MGE (1)). To ease the design of blockgenerators a tool was developed that allows a fast and flexible procedural assembly of basic cells (SILCO).

SILCO consists of a set of PASCAL procedures which allow the relative placement of basic cells (leafcells). These leafcells can be designed either using a polygon based layout editor for high density, either cells made with a symbolic layout editor or procedural leafcells (CAMELEON (1)). Dedicated channel and riverrouting are supported by this environment. Finally some example where this proposed methodology is used are presented.

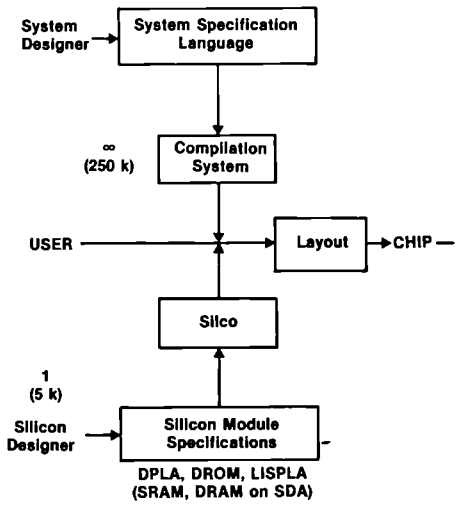


FIG. 1
MEET-IN-THE-MIDDLE DESIGN STRATEGY

2. FLOORPLANNING

2.1. Automatic routing

The floorplan methodology presented requires the availability of a block router which is tightly integrated with the rest of the CAD environment. The most important requirement is this router can be used almost without increasing the area compared to manual routing. An easy access to the database is also a very important feature. The requirements which should be met by a good router are listed in Table 1. Within ESPRIT97 we evaluated these criteria for the router of SDA (2). The main results of this evaluation are shown in Table 2. Besides quite good routing results compared to manual routing, the easy access to the database via the SKILL language made this router an appropriate tool to explore floorplanning methodologies.

2.2. Floorplanning Methodology (fig. 2)

Floorplanning today is usually restricted to making an inaccurate drawing which is part of a feasibility report. After the layout of most of the subblocks is finished one starts to think again about the floorplan. But then it is often too late and redesign of some blocks might be necessary. Making a good floorplan is not an easy job especially if one wants to estimate the effect of the routing. Not taking into account the effect of the toplevel routing, often results in considerable area increase after realisation of the sub-blocks. This increase is not only caused by the routing area but especially by the fact that the floorplan becomes inoptimal. Redesign of the sub-blocks in this stage of the design to fit better in the floorplan requires a major effort.

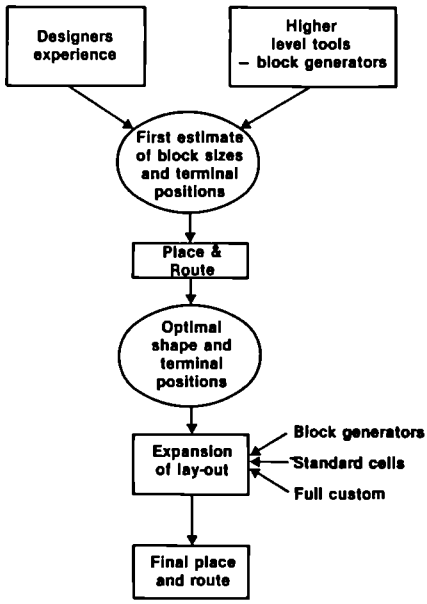


FIG. 2
FLOORPLAN METHODOLOGY

In our methodology the first step is to get an estimate of the block sizes and terminal positions of the different sub-blocks. This information is supplied by the designer or the higher level tools. The designer can enter this information either using the graphical editor either using the floorplan input file. This file allows the user to give the dimensions of each block, the pin positions and the nets these pins are connected with. The floorplan input file is read by a dedicated SKILL procedure which generates automatically outlines and terminals. If the designer has no notion were the input and output pins will come (e.g. for a standard cell block) a random placement is chosen by the SKILL procedure. If there exists a blockgenerator, exact dimensions and pinlocations can be entered automatically. The netlist is extracted from the toplevel schematic or has to be supplied in the floorplan input file.

The designer can now start the placement of the different blocks. As guidelines for the placement he can use his own knowledge of the functional block partitioning and the airlines which indicate which blocks are strongly interconnected. After this initial placement a first routing pass can be executed. An optimal placement is obtained by comparing the results after routing of the different placements. If the dimensions of the blocks seems to lead to inoptimal placements and routings one can change the shape of the blocks (e.g. the standard cell blocks) to fit better into the floorplan. Once a global placement is found that is thought to be optimal one can start a local optimization by moving the terminal positions. This is of course not possible for all blocks. These block dimensions and terminal positions are supplied as requirements for the layout people. Finally the floorplan is gradually refined as the exact layout of the different sub-blocks becomes available.

There are some clear advantages to this methodology. First of all there is an area saving since placement, routing and layout of the subblocks are part of the optimization process. By doing the routing with an automatic routing program considerable time savings are obtained. Even more important is the reduction of the elapse time since placement and routing are completed together with the layout of the subblocks.

3. BLOCKLAYOUT

Three different strategies for blocklayout are used. All three of them will be discussed in the following paragraphs.

3.1. Handcrafted Layout

This way of working is at this moment only used to develop analog blocks. Most of Alcatels applications are mixed digital and analog. For this purpose well known polygon pushers are used.

3.2. Cellbased layout

A lot of applications include what is usually called random logic. For these parts of a design a cell based technique is the most efficient way to go. Our past experience proved that when people want to layout such blocks by hand they always use rows of cells with the interconnections done in the channels between them.

A natural extension of this method is the use of standard cells. Here the placement in rows and the interconnection is done by a standard cell placement and routing program (2). This methodology for random logic parts of a chip is efficient both in area and in design time.

The development of timing models for a simulator - a timeconsuming job if one wants to do this properly - is a work which has to be done only once. Documentation is much easier. Experience with this strategy proved that this delivers designs with to timing errors, whereas this proved to be one of the most occuring problems with chips in the past.

As far as area is concerned previous designs proved that for a 2-micron 2 metal technology typical densities of 800 to 900 transistors/mm2 can be achieved. Cell transistor densities are typically 2000 transistors/mm2.

The last advantage of this strategy is the flexibility one gets with the aspect ratio. This is especially useful for our floorplanning strategy, where it is possible that the desired aspect ratio changes slightly during the design phase. This because the size of other blocks may change. For example the size of an analog block can have been underestimated. Or the number of instructions in a ROM can haven been overestimated.

4. MODULEGENERATORS

4.1. Quality Criteria

Different criteria are used to determine the quality of a module generator and of a module generation strategy.

One of them is the number of situations you can use your generator in. This can be done with parameters : e.g. the number of bytes in a RAM, the programming of a PLA, the size of a counter. A step further is technology independence. In this case a generator can be used as such, even when your design rules change.

Another one is the quality of the generated blocks : transistor density, speed.

A criterion which has been greatly been overlooked in the past is the ease of development of a module generator. We feel it is extremely important that module development can be done very fast. Most strategies of the past proved that actual development and full debugging of a module was extremely timeconsuming. Even then, e.g. a PLA could not be used in all situations. In the case of a PLA the basic design leads to small and slow PLA's or big and fast PLA's. Developing one type, which can handle all cases is impossible.

4.2. Strategies

Most strategies divide the problem into two subproblems : generation of basic cells; and a cell composition problem. Within the ESPRIT97 project both problems have been addressed.

4.2.1. Cell Generation

Three approaches are possible for cell generation. First one can work with fixed libraries. The libraries have to be redesigned for different technologies. The cells can also be designed with a symbolic layout system. At this time one has to regenerate the libraries when a new technology is used. This takes typically a few days, where the redesign of the cells takes typically a few months. This technique has been used within ESPRIT97 with the CAMELEON symbolic layout environment.

One can also design the cells with a procedural technique. This has been investigated with the development of a layout generator within the PLASCO environment. This technique proved to be extremely slow in development and to take a lot of time to debug. Therefore this had to be abandonned.

4.2.2. Cell Composition (fig.3)

Alcatel-BTM developed an environment for cell-composition. The designer of a module generator is free to use a cell approach he feels is the most suited to his needs. At Alcatel-BTM both generators with handcrafted - to achieve highest layout densities - and with symbolic cells - to achieve easy transfer of technology - have been developed. The generators are technology independent.

The composition-environment consists of a set of basic PASCAL routines, which build or read a data structure. This enables the user to write a 'composer' with high level calls. The advantage of PASCAL is that one has the full power of a proven programming language to develop generators. This allows the user to do easy manipulations of input files, the use of if-then-else while-do constructs. This also makes the environment easily extendable.

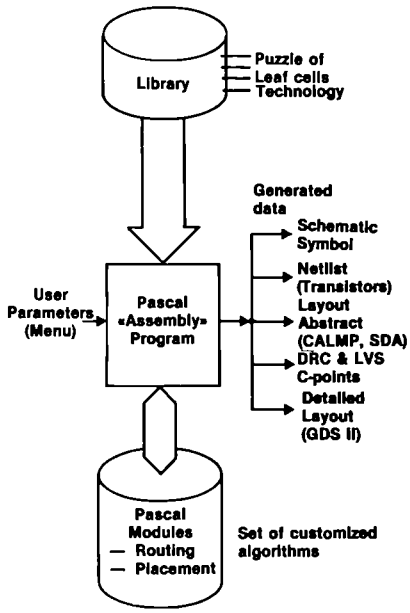


FIG. 3
SILCO : SILICON COMPILER ENVIRONMENT

The cell size information is available in the cell library. The user - module writer - has not to bother about this. In a cell the notion of terminal is also supported. Again the user can work with terminal names without bothering about the exact location of these terminals.

Below the key functionality is summarized :

Relative placement of cells. For instance one can specify that a cell has to be placed left to another one, or that the lowerleft and lowerright corners of two cells have to abut.

Database query functions : ask for location of a cell, of a cellcorner of a terminal on a cell.

Wiring functions : Connect terminal 1 with terminal 2 with a certain type of wire : for example a hookform wire. Riverrouters, dedicated channelrouters.

Technology information is present in a file read during initialisation. This is necessary to communicate to the environment the necessary parameters for the routing procedures. The minimal widths of wires etc...

ROUTER MAIN FEATURES

Automatic standard cell placement
Rectilinear blocks
Probing to help placement
Automatic macro cell placement
Automatic channel generation
Manual channel generation
Obstructions on layer per layer basis
Prewiring
Manually specifying global routing phase net per net basis
Probing after global routing
Priorities on nets
Equipotential terminals
Class of terminals in standard cells
Power and ground routing with special routing style
Routing nets on non minimal width
Current tapering
Hierarchy
Gridless routing
45 degrees routing
DRC correct
Staggered contacts at channel ends
Backtrack in design

TABLE I :
REQUIRED ROUTING FEATURES

BENCHMARK RESULTS

- Chip 1 (full-custom) :

  70 blocks, 300 nets, 80 K Tors, 55 mm².
  Automatic P&R → + 7 % Area versus handcrafted.
  Overall transistor density 1,400 Tors/mm²
  (2 μ - CMOS).
  Routing time : 2, 3 weeks.
  Number of iterations : 25.
  Power/ground were «Prewired», auto-routed with variable width.

- Chip 2, chip 3 (pseudo-custom) :

  Block based designs (module generators, analog ,full-custom & standard cells).

  17 block, 330 nets.
  30 % routing area (versus core blocks).
  40 mm², 30 K Tors.
  4 iterations.

  15 blocks, 300 nets.
  50 % routing area.
  20 mm², 13 K Tors.
  8 iterations.

TABLE II

4.2.3. Integration

The SILCO environment is integrated in a whole design environment. This means that more than just layout is provided. At the same time a netlist and a simulation model are provided with the correct delays included. Other outputs are also generated : for example a symbol to use it in a schematic entry program. In this way one can easily access the generator in the complete design cycle and not just during the layout phase.

5. EXAMPLES

5.1. MIP

The new generation of analog telephone sets will include more and more advanced features (last number redial, 10-number memory, on-hook dialling,...). The MIP (Maximum Integrated Phone) is a single CMOS chip which allows to make telephone sets providing such features. The MIP contains both analog and digital circuits. The RAM which, used for the last number redial and a storage possibility for 10 additional numbers of 16 digits, is implemented using the SILCO generator. The toplevel routing is done manually. The MIP die size is 16.9 mm**2 in 2.4 CMOS technology (fig.4).

5.2. SIC

The SIC offers a transmission system according to the CCITT I.430 for the 4 wire S-interface (ISDN standard) used in following configurations : Network Termination, Terminal Equipment, Trunk module and subscriber Line Module. The chip is designed in a 2 micron (shrinkable to 1.5 micron) CMOS double metal technology. The chip is about 20 mm**2 and contains 12k transistors. The chip contains both analog and digital circuits. Some of the blocks were designed using a standard cell approach. The PLA's were automatically generated with SILCO. The toplevel routing and floorplanning were done using the SDA router according to the mentioned methodology (fig.5).

5.3. MSRA

The MSRA is a Multi Standard Rate Adaptor designed for use in the Integrated Digital Services Digital Network (ISDN). It will enable building multi-standard terminal adaptors. The chip is 45 mm**2 and contains 35k transistors. For the technology we refer to the SIC. We could define this chip as a hierarchical standard cell chip. The different PLA's and the static RAM were generated using SILCO. (fig.6).

6. CONCLUSIONS

Alcatel-BTM partnership within ESPRIT97 project has resulted in a workable approach for a flexible compilation of structured Block design to silicon. The methodology achieves the right realistic trade-off between the actual requirements for higher productivity of secure designs and the presently available CAD technology for automatic translation towards the mask geometries.

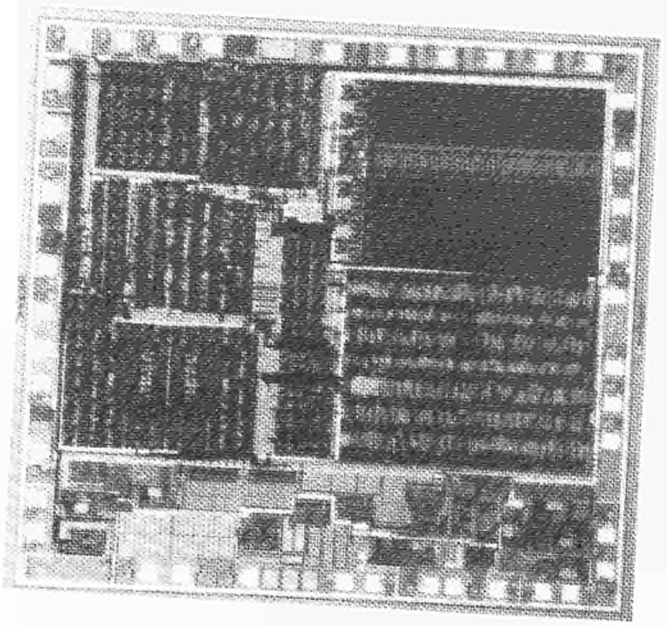


fig 4 MIP

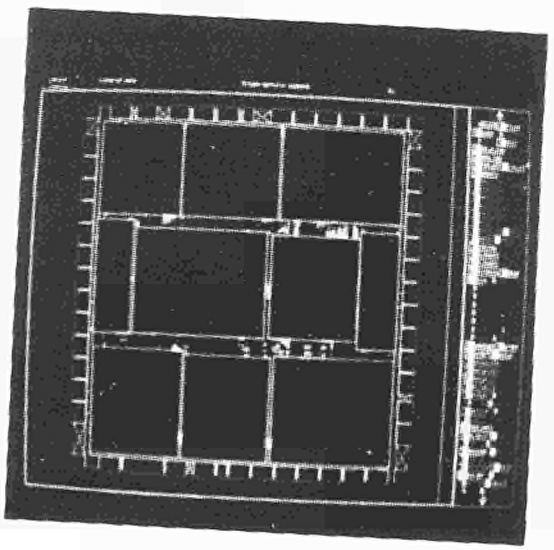


fig 5 MSRA

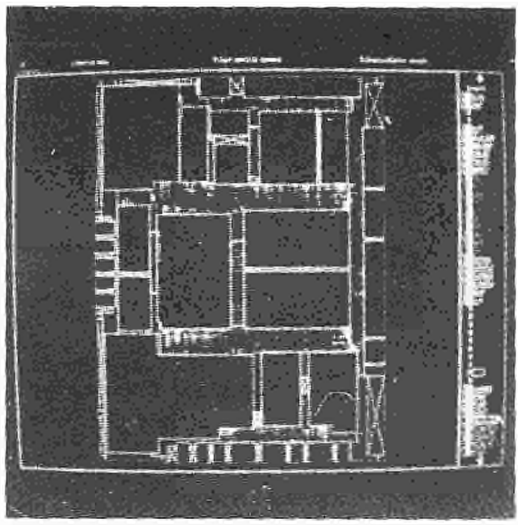


fig 6 SIC

The environment communicates with ESPRIT97 architectural synthesis with tools via the meet in the middle plug. Both architectural and geometrical synthesis realise true silicon compilation, the real achievement of ESPRIT97. At the layout side still technology independence remains a challenging unsolved problem.

7. REFERENCES

(1) L. Rijnders, A. Demaree, H. Deman "CAMELEON version 1.3 userguide" November 14, 1986 IMEC Kapeldreef 75 3030 Leuven.

(2) SDA Systems "Place and Route reference manual", march 20,1987.

Project No. 1058

Open System Architecture of an Interactive CAD Environment for Parameterized VLSI Modules.*

L.Claesen, Ph.Reynaert[†] G.Schrooten[‡] J.Cockx[§]
I.Bolsens, H.De Man,[¶] R.Severyns, P.Six, E.Vanden Meersch

IMEC, Kapeldreef 75, B-3030 Leuven, Belgium, (phone 32/16/281203)

Abstract

This paper describes an open system architecture for *interactive and communicating* CAD programs. Within the ESPRIT-1058 project it is the framework for knowledge based and high performance design tools of modular VLSI designs. The goal of the system architecture is to provide a direct *interaction* and *feedback* between the primary design tools (schematics editors, symbolic layout editors, module generators etc.) and intelligent verification tools (electrical debuging, timing verification, simulation etc.). The tools run in *parallel* and have *bidirectional interactive communication* by pointing to objects in primary design inputs and passing information to verification tools and by allowing verification tools to highlight objects in primary design editors.

Keywords: CAD, Open frameworks, CAD architecture, user interfaces, tool integration.

1 Introduction

Current VLSI design is continuously shifting towards more and more complex system design. In order to be able to cope with the emerging complexities, new design methodologies are ultimately required. One consequence of this emerging complexity challenge is research towards the application of silicon compilation techniques, that start from high level behavioral specifications. Techniques with general application possibilities are still far away. However,

*Work performed within the scope of the ESPRIT-1058 project: "Knowledge Based Design Assistant for Modular VLSI Design". Partners: IMEC Leuven Belgium, INESC Lisbon Portugal, Philips Eindhoven The Netherlands, Silvar-Lisco Leuven Belgium

[†]Silvar-Lisco, Abdijstraat 34, B-3030 Leuven, Belgium. (phone 32/16/200016)

[‡]Philips Research Labs. Eindhoven, Prof. Holstlaan, NL-5600 JA Eindhoven, Netherlands. (phone 31/40/743897)

[§]with Silvar-Lisco

[¶]Professor at Kath. Univ. Leuven Belgium

when aiming at dedicated applications and known target architectures, operational systems demonstrate the feasibility of automatic synthesis from high level specifications [1,2]

The CATHEDRAL-II silicon compiler [2] as developed in the ESPRIT-97 project is aimed at digital signal processing systems in the broad sense in a multiprocessor target architecture [4]. The general structure of CATHEDRAL-II is shown in figure 1.

The basic strategy adopted in the CATHEDRAL-II system is the so-called *"meet-in-the-middle"* design methodology [3]. This is a methodology that, due to the high complexity system requirements, is adopted more and more in VLSI system design in order to be able to manage the design complexity.

The meet-in-the-middle design methodology is schematically indicated in figure 2. This figure shows the abstraction levels from high level specifications to transistor implementations. In this methodology, the VLSI design activity is split in two parts: first the system design activity, which is done by system designers and secondly the silicon design activity which is done by silicon and software oriented designers. The separation between the two is at the level of modules. Examples of modules are generic blocks like Multipliers, Dividers, ALU's, ACU's etc.. In order that these modules can often be reused, the modules must be *parameterizable* to allow customization to specific system design needs.

A key point in the success of the meet-in-the-middle strategy is that system designers can make *abstractions* of the implementation details (such as circuit level details etc...) of the parameterized modules. Therefore these modules, together with the procedural descriptions of the required views must be *created, verified and characterized* very well in advance by silicon design experts (i.e. before these modules are actually used by system designers). In the ESPRIT-97 project the (automatic) system synthesis activities for multi-processor signal processing applications in terms of parameterizable modules is under development.

In the ESPRIT-1058 project, the meet-in-the-middle strategy is supported for the activities of interactive knowledge based verification [7]. In the meet-in-the-middle strategy, this verification is concentrated very much on the *creation*, *verification* and *characterization* of the flexible silicon modules, because they must afterwards be usable in a reliable and predictable way. Modules are described in a parameterizable way using LISP-based module generators that define how to compose them of leaf cells designed by means of symbolic layout [18,17,15].

For the correctness verification of these modules, knowledge based approaches are used [8] for verifying design soundness with respect to composition rules, electrical rules, electrical debuging. Other tools have been developed for the verification of timing correctness [9]. The major goal of these knowledge based analysis tools is to try to reduce the need for extensive circuit simulations. Circuit simulation will however always remain important. Therefore circuit simulation techniques for large circuit modules are being developed based on waveform relaxation techniques [13] and explicit integration techniques [14]. New algorithms are under development for exploiting parallellism in these circuit simulation techniques for implementation on parallel processing hardware.

One of the major bottlenecks in the efficient application of these verification techniques is the interactivity between verification and simulation tools and the designer.

The current design practice is that CAD tools are run one at a time and that communi-

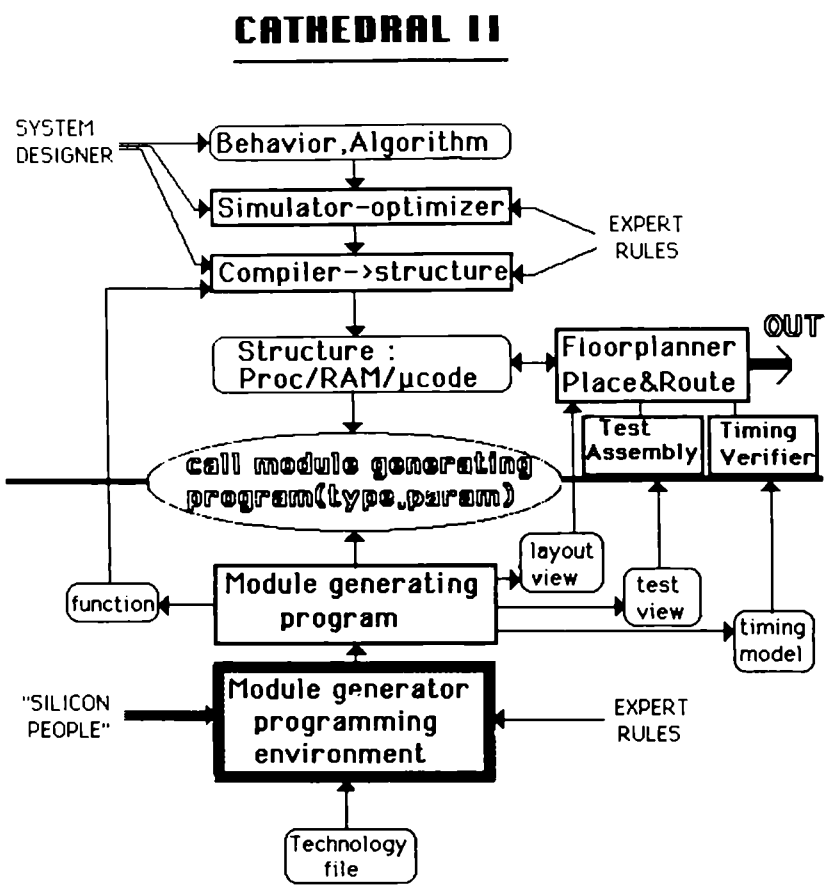


Figure 1: Overview of the CATHEDRAL-II silicon compiler.

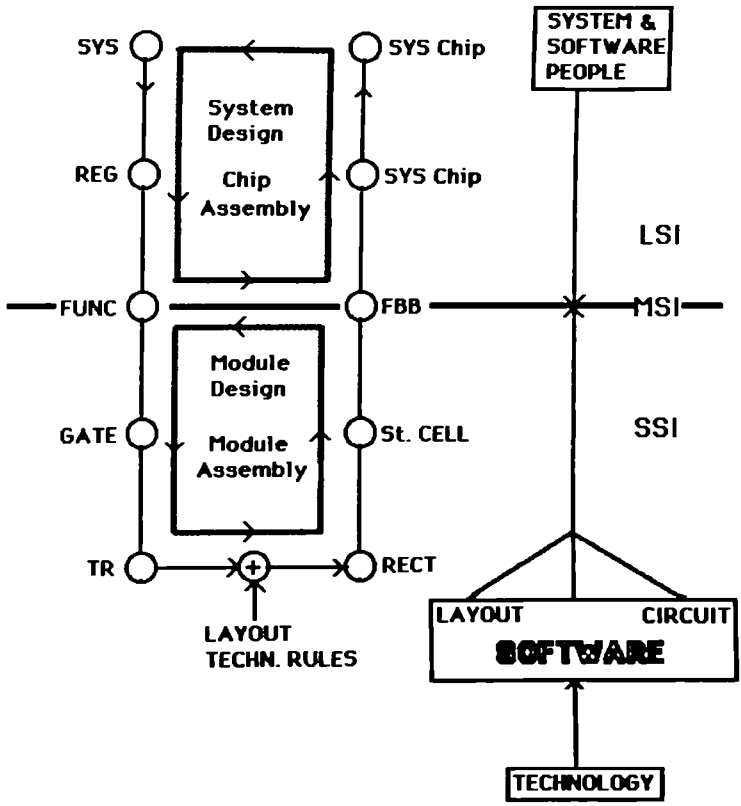


Figure 2: Meet-in-the-middle design strategy

cation of information is via files and cross-reference lists. This is extremely time consuming in a verification or debuging phase where the feedback between design definition and design debuging is currently taking most of the designers time. An other disadvantage of the current CAD tools is that they often require different formats for representing netlist information, which necessitates the use of cross reference lists and makes it harder for the designer to relate information from a simulator or a timing verifier to the original information.

In order to allow for a much faster feedback between knowledge based verification tools and the module designer, an open and interactive system architecture is under development as part of ESPRIT-1058. This is the main subject of this paper.

In the next section an outline of the research work in ESPRIT-1058 is given. Thereafter the following sections describe the part in ESPRIT-1058 that deals with the open and interactive system architecture.

2 Overview of the ESPRIT-1058 project

In this section the major research topics in the ESPRIT-1058 project entitled "Knowledge Based Design Assistant for Modular VLSI Design" are presented.. The goal of the project is the realization of an interactive knowledge based system for the verification of the electrical, behavioral and timing correctness of flexible VLSI modules.

The research topics are:

- Open system architecture and user interface.
- Accurate timing verification.
- Knowledge based electrical verification.
- Acceleration techniques for accurate circuit simulation.
- Symbolic layout and extraction.

In the next subsections these topics are summarized.

2.1 Topic: Open system architecture and user interface.

The topic open system architecture is the subject of this paper. It allows for an easy integration of knowledge based verification tools. It provides an interactive communication among the designer and tools running concurrently. It is the basis for an efficient interface between a designer and knowledge based verification tools.

2.2 Topic: Accurate timing verification.

SLOCOP [9] is an accurate timing verifier for synchronous digital MOS circuits. Design style dependent circuit partitioning can be expressed in the LEXTOC language[8]. Accurate delay modeling of subnetworks is achieved by using circuit simulation with the SIMMY module. In the critical path analysis, *false paths [10] are automatically removed.*

A *hierarchical delay modeling technique* allows for the delay characterization of silicon modules. This is a requirement for efficient timing verification on floor planning level in CATHEDRAL-II.

In SLOCOP a new critical path method determines the longest delay path. All existing timing verifiers for MOS [11,12] have problems with the generation of *false paths.* In SLOCOP this problem is solved by taking into account the logical consistency of the subnetworks in order to avoid the indication of false paths [9,10].

The timing verifier can operate interactively and gives direct feedback by highlighting critical paths to the schematics editor or to the CAMELEON [15] symbolic layout system, by using the SPI interface [21].

2.3 Topic: Knowledge based electrical verification.

In order to decrease the need for circuit simulation, the DIALOG [8] electrical and topological verification system is developed.

DIALOG allows to formulate knowledge about a circuit as production rules, using the LEXTOC language. The knowledge base contains rules that indicate violations against good design practice. The system allows to verify : the correctness of topology, the creation of good logic levels, checking for DC-sneak paths, the obedience to the clocking protocol, logic decompilation etc.. Back annotation to the schematics editor is possible via the system architecture presented in this paper.

Rule-based activation of the SIMMY circuit simulation module, together with appropriate stimuli on critical parts in the design is possible as well as the automatic evaluation of the simulator outputs.

The system provides a *rule interpreter* that allows for an *explanation facility* when possible design bugs are discovered. Modules with up to 18K transistors have been verified on a LISP machine in 30 minutes.

In order to allow for an adequate use of such a knowledge based debuging tool a strong and frequent interaction between the tool and the designer is required via the primary design inputs such as schematics or layouts. This allows explanation facilities on possible bugs, such that the designer can make the appropriate decisions *fast.* Such an interactivity is not possible in a *general way* in existing tools.

The DIALOG environment also supports knowledge based verification of incremental evolving designs. This requires a *fast interactive* feedback between primary design entry tools and

the DIALOG verification tool.

2.4 Topic: Acceleration techniques for circuit simulation.

Classical approaches for circuit simulation as used in SPICE are limited to a few hundred transistors due to limitations in simulation speed and available computer resources. Even small VLSI submodules consist of much more transistors. Therefore other simulation approaches, that are much faster and allow for larger circuits, with the same accuracy as SPICE are under development.

A first approach is waveform based circuit simulation[13] as implemented in the SWAN program. This program allows for simulation of 4000 transistor circuits in a few minutes cpu time with the same accuracy as SPICE. Current effort in the project is devoted to accelerate circuit simulation by developing new algorithms suited for parallel processing hardware.

A second technique is under development at INESC [14]. This method uses a new integration scheme based on the discretization of the voltage axis instead of the time axis. This technique leads to a very efficient event driven algorithm that takes full advantage of latency, that is pervasive in digital integrated circuits, without the overhead associated with static network decompositions.

Fast interactivity of these circuit simulation tools with designers is possible via the system architecture presented here.

2.5 Topic: Symbolic layout and extraction.

Symbolic layout [15,16] is used to design leaf cells in a layout rule independent way in the CATHEDRAL-II system. The CAMELEON [15] system is coupled to the MGE [17] module generator. Extraction directly from symbolic layout is necessary in the project to allow for the direct coupling to the knowledge based verification tools. This coupling is achieved through the system architecture that is described in this paper.

The research concentrates around modeling parasitic effects due to submicron litography in VLSI.

3 Open system architecture for interactive CAD.

Much attention is currently put on the issue of open system architectures for electronic CAD programs [5] and prototypes are under development [6]. Irrespective of *sound-alike sales-talk claims*, there are currently no *open* CAD environments commercially available that allow to *imbed* strongly interactive CAD tools in an easy way.

This paper explains the outline for an *open architecture* allowing for *bidirectional* communication between interactive CAD tools running in parallel [19]. It is conceived in such a way

that it is possible to interactively communicate between user-interface tools and simulation and verification tools by user-pointing and screen highlighting of structural objects. The architecture is set up in such a way that individual tools can be developed rather independently. (or that independently developed tools can be integrated with minor work). The system architecture majorly integrates around the concept of structural data that is communicated between tools. In order to integrate existing tools easily, they are not very much constrained. Tools are allowed to have their own "primary data". It is only required that the tools are also able to generate the structure information in a standard way from this "primary data". The fact that there are not too much constraints on the tools themselves allows that existing CAD tools can be integrated without too much difficulty. This fact together with a good definition of *procedural interfaces* for communicating to other tools contributes to the aspect of an *open system architecture*.

Important in this architecture description is the philosophy of allowing CAD tools to communicate interactively to the user via the primary input descriptions. In this way a much faster feedback is achieved to the designer and also a more efficient design cycle is achieved.

In this system the tools are organized around the concepts of:

Common Database Organization . This does not imply the use of one database or the use of the same formats. (Nevertheless it is encouraged). This is to access the files as a whole in a standard way, not the contents.

Communicating application programs: In the underlying system it is required that two or more programs can communicate with one another in a *bidirectional* way. This is necessary to allow *highlighting* of and *pointing* to structural items. In order to achieve this a structure communication standard SPI is used [21].

Uniformization of structure data: By structural data is assumed data concerning:

cells . These items are also known by the database interface.

components or instances of cells.

terminals of cells

terminals of components

nets

as well as some general attributes that can be associated with each of the structural entities described above. *Parameters* are a special type of attributes that are associated with cells (formal parameters) and components (actual parameters).

Databases used by application programs: In order to be able to put as few constraints on application programs as possible, the general rule is that *each application tool is allowed to have its own formats for databases or data files.* The requirement is only that the designer, or some design tools have to indicate to the DMS database interface which view (generated by which tool) contains the primary data for the cell from which the structural data can be derived. The structural data is standardized. This data should be extracted from the own data structures of structure generation tools and passed in a standardized way to tools that act on the structural data. The fact that the other databases are not that constrained is advantageous in order to incorporate existing CAD tools in the system. It also puts less constraints on new tools that have to be developed.

Primary Data Assumption. In the module design environment it is assumed that the designer declares for each cell which tool defines the correct primary data. (This is done through the DMS.). In the database this is reflected by the fact that for each cell there must be information which tool has defined the primary data for that cell. The primary data corresponds in fact to the information that the designer has initially entered as specification of the design. An example is symbolic layout [15]. Secondary data is then for example the physical layout generated from the symbolic layout. Other "so called" primary data may also exist but is not considered as primary in this text. An example is a cell with the symbolic layout which as declared as primary data by the designer, and of which also a schematic has been made. In this case the schematic data is "so called" primary data but not "the" primary data. The meaning is that the schematic is in this case not the primary definition for the structure of the cell. The structure as represented by the schematic can be compared to the primary structure by a netlist comparison program such as for example WOMBAT [22]. The DMS design management system has to be aware of which view primarily defines the structure for each cell, because this information will be used by preference for further verification work.

Implementation details: All operations of CAD programs which are communicating together via SPI are considered to be *synchronized.* This means that only one tool will be activated at a specific time. Other processes are then in a waiting state.

4 Tool summary and interaction.

Figure 3 gives the major communication. The notational conventions used are indicated in the figure. The functionality of the system is of prime importance. An implementation of the system can be done over one or more processes running on one or more machines. The DTM *data & tool manager* is the main controller and selects active projects (or cells) and activates CAD-tools. The DMS *design management system* selects the database files from the operating system file system. The SPI *structure procedural interface* is the *structure communication bus* between the CAD tools.

Now follows a further discussion of the functionality of the individual modules depicted in fig.3.

4.1 DTM : Data & Tool Manager.

The Data and Tool Manager is the main process controlling the global CAD activity. This tool allows to define the *current project(s)* on which actions are to be performed. Then the CAD tools such as editors, verifiers and simulators can be called. DTM is aware of the available tools and of the available cells and it controls the switch box function around the SPI procedures such that data is routed from the appropriate structure generation tools to the appropriate structure receiver tools.

The DTM consists of the main control loop with the user. The designer can:

- either indicate projects to be used by the programs.

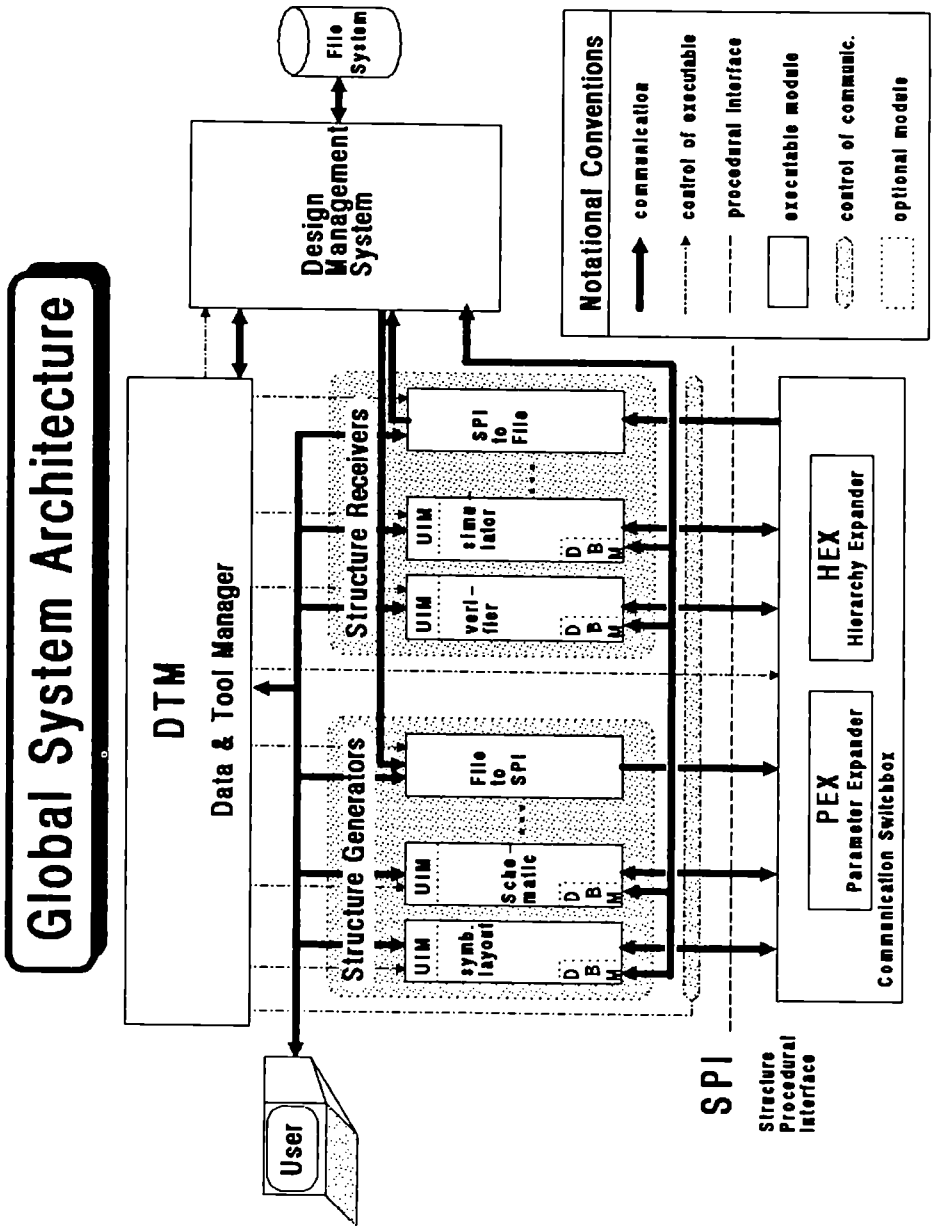


Figure 3: System Architecture

- or control the execution of application programs and inter process communication between application programs.

The functionality of the DTM could be extended on top of the functionality of existing "monitor" programs encountered in several CAD systems. The extension required by this architecture is the ability to manage several *communicating* CAD-tools, instead of only managing one tool at a time.

4.2 Application programs.

The application programs can be classified in two major groups:

1. Structure generation programs.
2. Structure receiving programs.

The two classes of application programs are further discussed in the next subsections. One program can also belong to the two classes. An example is a preprocessor program that takes structure and transforms it into other structure.

4.2.1 Structure generation programs.

The structure generation programs consist of the application programs which have a direct graphical interaction with the user. These programs are for the most part editor-like and communicate in a graphic way. These programs are majorly used to provide *primary input* of design data to the system.

Examples of these foreground application programs are:

- schematics editor.
- layout editor.
- sticks editor.
- textual editor.
- algorithmic floorplanner, parametric module generator.

4.2.2 Structure receiving programs.

The structure receiving programs are the programs that do not have a direct input capability for primary design data.

Examples are:

- simulators [13,14]
- timing verification programs such as SLOCOP [9]
- electrical verification programs such as DIALOG [8].
- floorplanner, block place & route
- place & route programs.

The major reason behind this subdivision of application programs, is that it is intended that there will be a strong interaction between the application programs and the designer/user of these programs. Therefore structure generation and structure receiving programs must be interlinked very closely. This must allow the designer to pass information on his design from the structure generation programs to the structure receiving application programs and get feedback in the reverse way by user pointing to structural items and highlighting. In this way two tools can act as one global integrated tool.

4.3 SPI : Structure procedural interface.

The structure procedural interface [21] is a switch box and a standard way to communicate *in a bidirectional way* structure information (cells, components, terminals, nets and attributes on these) between structure databases, editors (for interactive highlighting) and verification and simulation programs.

As part of the structure procedural interface there are two expanders available HEX and PEX. A hierarchy expander (HEX) which removes hierarchy in a hierarchical description and a parameter expander (PEX) which removes structural parameters in a parameterized structure description. This is necessary if verification or simulation tools, that only understand flattened information, want to communicate with one of the user-interface editors by pointing and highlighting.

SPI is a standard *procedural interface* on structural data, suited for programs. The same interface can be used to communicate between editor and files, in which all structured information is stored, and between these files and verification and simulation programs, but only in one direction (no highlight). Besides SPI, there is also a directly derived textual representation, to be used for archival purposes and for the communication with tools where no coupling via procedures is possible. Conversion between these direct textual representations in files and SPI procedures are provided by tools `File to SPI` and `SPI to File` in figure 3.

Dual to the SPI procedures there is also a designer oriented textual representation HILARICS. The coupling between the *designer oriented textual representation* and the application programs will be provided by a HILARICS to SPI compiler. In this way application tools should only use the SPI procedural interface. HILARICS is then "the" primary input.

4.4 HEX : Hierarchy expander.

The hierarchy expander provides a bidirectional link between structural data on level 1 (hierarchical) to structural data on level 0 (flat). To provide the possibility for bidirectional communication, local storage of datastructures is necessary to keep the links between structure data on level 0 and structure data on level 1. The user determines via the DTM and the application tools which data has to be expanded.

4.5 PEX : Parameter expander.

The PEX module is similar in functionality to the HEX module. It provides a bidirectional translation between tools that generate structural data on level 2 (parameterized) and structural data on level 1. The same observations as made for the HEX module also hold here.

4.6 DMS : Design Management System..

The Design Management System performs a mapping of design objects such as *designs*, *cells*, *views* and *versions* in the file system. A design object is the basic building block to be used at the design management level (schematic, layout, behavioral model of a cell, parameterized description of layout, ...). It returns a unique identification of a file on which the other tools can act. The DMS is not aware of the contents of the design objects. The contents of the design objects is determined by the tools that act on it.

There are many alternative implementations possible. The specific implementation is however hidden behind the DMS-access procedures. Depending on the needs in a specific design environment, the implementation can have very extended features and facilities (such as protections etc.) or can be very elementary. Within the scope of the system architecture, the functionality of handling basic design objects is the major requirement.

The DMS also knows for each cell which view defines the *primary data* that determine the structural data for the view under consideration. This information with each cell is required in order that the DTM together with the SPI can decide which tool has to be called for each level of cell, in order to generate the required structural information for the structure receiving tool.

In the DMS also an attribute with *the usage level* of each cell should be indicated. The *usage level* is for example `circuit level`. This means that the SPI procedures should call starting from the top cell, structure generating tools until the circuit level (transistors, capacitors etc.). This information could be entered for example via the DTM.

4.7 UIM User Interface Module

The UIM provides a consistent user interface to be used for all tools. It is not essential for the system architecture as such. However if two tools are being integrated, a consistent user

interface is advantageous: menus, windows, highlighting, selecting, message handling are done in a consistent way. Therefore a common UIM is encouraged because it contributes to the openness of the system architecture.

4.8 The Linker.

The linker[1] generates the descriptions of cells that system design tools need for instances of the cells with actual values for parameters. This information is derived from the parameterized descriptions by either executing programs or accessing the DMS.

The linker [20] is the software concept that supports the "meet in the middle" design strategy in a standardized way. In this way the linker provides an abstract interface between parameterized descriptions of cells and system design tools that require the generation of properties for module instances, with specific actual values for its parameters. The functionality of the linker is in some sense orthogonal to the functionality of the system architecture as shown here: The linker provides an interface and the system architecture concentrates on interactively communicating programs.

The linker is used in application programs as a software module. The linker has a coupling to the DMS in order to get access to parameterized descriptions. It also is able to start executable programs.

5 Description of a sample session.

To understand better the operation of the system architecture, an informal description of a sample session is given. Notice that most of the actions are activated automatically in the system.

5.1 Sample design situation description.

Assume that a design of a multiplier has been made and that an instance of an 8 by 8 multiplier is available in the database. Assume that the cells are organized by the DMS as shown in figure 4. The 8 by 8 multiplier is represented in the DMS as the cell **mult8x8**. Assume that this cell is constructed using components from cells **fulladd, booth** and **mux**. The cell **mult8x8** is generated by the `MGE` module generation program [17]. Assume also that the designer has defined that the structure is primarily defined by `MGE` for the top cell **mult8x8**. The structure of the leaf cells is primarily defined by the symbolic layout program `CAMELEON`. Notice that the DMS is aware of which program primarily defines the structure.

Assume that the designer has indicated to the DMS the *usage levels* for the cells. In that sense the *circuit level* has been assigned to the cells **fulladd, booth** and **mux**. This is necessary afterwards for the SPI implementation to know untill what level the design has to

[1] not to be confused with a linker in software programs.

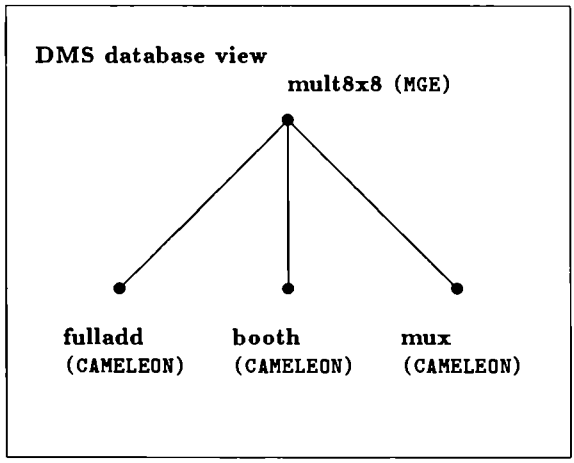


Figure 4: Sample database organization.

be passed to the application program by the SPI procedures. DIALOG [8], a circuit simulator like SPICE or SWAN [13], a timing verifier like SLOCOP [9] all require *circuit level.*

Suppose that the designer wants to perform interactively an electrical verification by using the DIALOG [8] program.

Via the DTM the designer can interact with the DMS. He can see what information is in the databases.

Interacting with stand-alone tools such as editing-only sessions is done in the normal way. This means that projects from the DMS are selected by the designer through the DTM and that the appropriate CAD tools are started in a stand-alone way.

In this way the information in the database for `MGE` or `CAMELEON` could for example be created.

The case where more tools are activated at the same time is described in the following sessions.

5.2 Initiation of a verification session.

Suppose the designer is at the point of checking the **mult8x8** design for electrical bugs. Therefore he first has to enter the DTM, (via a special monitor window on the screen). There the designer can browse trough the available projects managed by the DMS. He will than choose **mult8x8**. This in fact defines the top cell in the hierarchy. Together with one DTM the designer can start one or more CAD tools which will cooperate in *a synchronous way* under the direction of the DTM.

Now the designer can start one or more CAD tools, (each as separate processes and in separate windows).

In this design application the designer could start a window (process) with the module generator MGE and leave MGE in a waiting state controlled by the DTM. In the same way CAMELEON could be started in a window (process) and be left in a waiting state under control of DTM. Notice that in this example situation *two* different programs are started. These programs can be made active by the DTM to do editing or by the DTM and SPI to load a specific cell and generate structural data for the SPI interface.

Suppose that MGE and CAMELEON are both active and that the designer starts DIALOG from the DTM.

After this action DTM gives control to DIALOG for doing its initialization and for loading the information for the project at hand **mult8x8**. Therefore DIALOG gives the control back to the DTM, which will now activate the SPI module.

SPI will now recursively go trough the DMS design tree and activate each of the appropriate structure generation tools with the appropriate DMS-cell indication. Each of these tools will then for each of the cells call the appropriate SPI procedures to download the structural information to the SPI module.

In this case this means that the SPI will start from the top cell **mult8x8** and ask the DMS for the primary view. In this case this is `MGE`. SPI activates MGE with cell **mult8x8**. MGE then generates the structure information via the SPI calls. MGE keeps the information of this cell in its datastructure and goes back in a wait state and gives back the control to SPI. SPI then knows via the SPI calls of MGE which cells are used as components of **mult8x8**. Now SPI can again activate tools to generate the structure information for the cells of which it does not have the structure information yet. This is the case for the cells **fulladd**, **booth** and **mux**. SPI therefore goes through this list one by one and performs the following actions:

> It asks the DMS for the primary view of the cell at hand. First this will be **fulladd** with primary view `CAMELEON`. Then SPI activates CAMELEON with cell **fulladd**. CAMELEON will then ask the specific `CAMELEON` file in the database for cell **fulladd** through the DMS. It loads the information and passes the derived structural information to SPI. CAMELEON keeps this information and goes to the wait state again. Control goes back to SPI. SPI will than take the next cell (e.g. booth and repeat the same actions).

After this CAMELEON has been activated three times by SPI with three different cells. The information of these cells is each individually kept in CAMELEON datastructures and has been passed to the SPI module via SPI procedures.

Now all information is available in the SPI module. Because DIALOG only understands flat circuit descriptions, the HEX module in the SPI block will be used. In the HEX hierarchy expander all links between the hierarchical network and the expanded circuit are kept, in order to allow communication afterwards.

After the expansion all information in the SPI module is passed to DIALOG via SPI

procedures. After this action by SPI control is given to DTM, which forwards control to DIALOG which can start its verification work.

5.3 Interactive communication between editors and application program.

Now the case of a highlight will be described. The case of a user select operation is dual to the highlight.

Suppose that DIALOG wants to communicate a possible electrical bug to the user via the editors. Therefore DIALOG will mention in its own window a textual description of the kind of bug it has found.

DIALOG will now indicate to the SPI module which structural item(s) need to be highlighted in the editors.

> In the case of a highlight or a select operation the SPI will pass control to the user, so that the user can choose the components in which the highlights or selects have to be done. For the case of highlight, this means that after DIALOG has passed control to SPI, SPI will give control to the user and let him/her choose among the available (and possibly affected) components that he/she wants to see. This is necessary because highlights and/or selects can affect more cells in a design. It is most often even different in components of the same cell.

The control that SPI gives to the user is by a (hierarchical) table of the components used in the design.

During these highlight or select operations the designer can look at the primary data as has been entered in CAMELEON or MGE. Remember that all cells in this project **mult8x8** are available in the data structures of these structure generation tools. Remember also that SPI keeps al the hierarchical relations between the flattened circuit and the hierarchical circuit.

After the highlight or select operation, the user can indicate to the SPI process that the highlight or select operation is finished. Now SPI can give control back to DIALOG, which can do further verifications with possible highlights or selects.

After the DIALOG verification session finishes, the control is given back to the DTM.

6 Interfacing of tools in System Architecture.

The interfacing of the tools in the system architecture is illustrated in fig.5. In order to obtain a system that is as open as possible, and also in order to allow a gradual integration of tools in the architecture some interfaces are mandatory and others are optional.

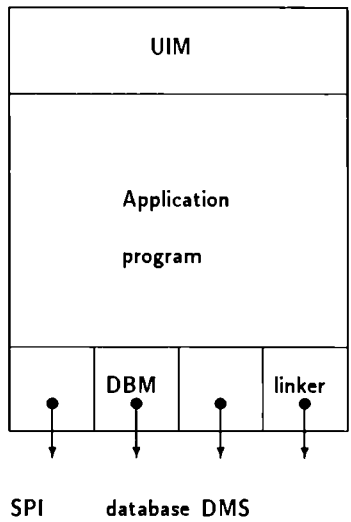


Figure 5: Integration of application program in the system architecture.

Mandatory interfaces . These interfaces are required if a bidirectional communication of the tool with an other tool is envisionned. It requires the interfacing to the following modules:

SPI structure procedural interface to communicate (either receive or generate structure) with other tools.

DMS Design Management System. to indicate which design objects are handled by the application program. This is required in order to know by the DTM what design objects are being handled by which programs.

Optional interfaces .

UIM User interface module to have a uniform appearance to the user.

DBM Database module to store the proprietary data of an application program on a design object.

Linker module : this is only of importance for system design tools that require information of parameterized cells.

In order to allow high lighting of structural items in more cells, tools need to be able to be activated with more cells at once. Otherwise if highlights or pointing has to be done, the tools need to be started per cell. This would require too much processes.

7 Summary.

In this paper, a *flexible* framework for *interactive* CAD tools has been presented. The emphasis is put on the *interactivity* of the CAD verification tools. This is required in order to be able to communicate in an interactive way with the *graphical* design tools a designer is confronted with. In this way the design cycle is shortened and verification tools can have closer communication with the designer. An example of such a tool is DIALOG [8] that can directly, while doing its analysis, indicate possible design errors in the schematics. Another representative analysis tool is the SLOCOP [9] timing verifier that can directly indicate critical delay paths in the schematics or symbolic layout by immediate highlighting, without stopping either the timing verifier or the schematics- or the symbolic layout editor. As less as possible constraints have been put on the tools themselves in order to be able to integrate already existing tools and tools obtained from outside with minor efforts.

References

[1] R.Jain, F.Catthoor e.a., "Custom Design of a VLSI PCM-FDM Transmultiplexer from System Specifications to Circuit Layout Using a Computer Aided Design System", *IEEE Journal of Solid-State Circuits*, February 1986, Volume SC-21, Number 1, pp.73-85.

[2] H.De Man, J.Rabaey, P.Six, L.Claesen, "CATHEDRAL-II: A Silicon Compiler for Digital Signal Processing", *IEEE Design & Test of Computers*, December 1986, Vol.3, No.6, pp.13-25.

[3] H.De Man, "Evolution of CAD tools towards third generation custom VLSI design", *Revue Phys. Appl. 22*, Vol. 22, January 1987, pp. 31-45.

[4] F.Catthoor e.a., "General Datapath Controller and Inter-communication Architectures for the Creation of a Dedicated Multi-Processor Environment", *IEEE ISCAS Conf.*, May 1986, pp. 730-731.

[5] K.H.Keller, "An Electronic Circuit CAD Framework", *Memorandum No. UCB/ERL M84/54, Ph.D Dissertation, University of California, Berkeley*, 6 July 1984.

[6] D.S. Harrison, P.Moore, R.L.Spickelmier, A.R.Newton, "Data Management and Graphics Editing in the Berkeley Design Environment", *IEEE International Conference on Computer-Aided Design ICCAD-86*, November 11-13, 1986 Santa Clara CA., pp.24-27.

[7] L.Claesen, H.De Man, I.Bolsens, W.De Rammelaere, D.Dumlugol, P.Lammens, P.Odent, R.Severyns, E.Vanden Meersch, "Electrical, Timing and Behavioral Verification in the Meet-in-the-Middle MOSVLSI Design Environment of CATHEDRAL-II", *Proceedings IEEE International Conference on Computer Design: VLSI in Computers & Processors, ICCD-87*, Port Chester, New York, Oct.5-Oct-8, 1987.

[8] H.De Man, I.Bolsens, E.Vanden Meersch, J.Van Cleynenbreughel, "DIALOG: An Expert debuging System for MOSVLSI Design", *IEEE Transactions on Computer Aided Design*, CAD-4, No.3, June 1985, pp. 303-311.

[9] E.Vanden Meersch, L.Claesen, H.De Man, "SLOCOP: A Timing Verification Tool for Synchronous CMOS Logic", *Proceedings European Solid State Circuits Conference*, ESSCIRC'86, Delft, September 16-18, 1986.

[10] J.Benkoski, E.Vanden Meersch, L.Claesen, H.De Man, "Efficient Algorithms for Solving the False Path Problem in Timing Verification", *Digest of technical papers IEEE International Conference on Computer-Aided Design ICCAD-87*, Santa Clara CA, November 9-12, 1987.

[11] J.K.Ousterhout, "A Switch-Level Timing Verifier for Digital MOS VLSI", *IEEE Transactions on Computer Aided Design*, Junly 1985.

[12] N.P.Jouppi, "Timing Analysis for nMOS VLSI", *Proceedings of the 20th Design Automation Conference*, pp.139-147, 1980.

[13] D.Dumlugol, P.Odent, J.Cockx, H.De Man, "The Segmented Waveform Relaxation Method for Mixed-Mode Switch-Electrical Simulation of digital MOS VLSI Circuits and its Hardware Acceleration on Parallel Computers", *Proceedings IEEE Conf. ICCAD'86*, Santa Clara, CA, Nov. 1986, pp. 84-87.

[14] L.M.Vidigal, S.R.Nassif, S.W.Director, "CINNAMON: Coupled Integration and Nodal Analysis of MOs Networks", *23rd Design Automation Conference*, June 29-July 2, 1986, pp.179-185.

[15] K.Croes, L.Rijnders, "CAMELEON: A Technology Independent Symbolic Layout System", *Internal Report IMEC, MR03KUL-7-B3-2*, January 1986.

[16] K.Croes et al.,: "CAMELEON, a process tolerant symbolic layout system", *Digest of technical papers ESSIRC-87*, Bad Soden, Germany.

[17] I.Vandeweerd, "Module Generation Environment: reference manual", *Internal Report ESPRIT1058/IMEC/6.86/D8619.*

[18] P.Six, L.Claesen, J.Rabaey, H.De Man, "An Intelligent Module Generator Environment", *Proceedings of 23rd Design Automation Conference*, Las Vegas, June 29-July 2, 1986, pp. 730-735.

[19] L.Claesen, Ph.Reynaert, G.Schrooten, "Open system architecture for communicating CAD tools", *Report ESPRIT1058/IMEC/12.86/D8652*, IMEC Leuven, Belgium.

[20] L.Claesen, Ph.Reynaert, G.Schrooten, "LINKER module", *Report ESPRIT1058/IMEC/12.86/D8653*, IMEC Leuven, Belgium.

[21] J.Cockx, Ph.Reynaert, "ESPRIT-1058: SPI Specification", *Report ESPRIT1058/SL/12.86/D8654*, IMEC Leuven, Belgium.

[22] R.L.Spickelmier, "Verification of Circuit Interconnectivity", *Report Electronics Research Laboratory*, Univ. of California Berkeley, June 1983.

Project No. 271

ADVICE: Automatic Design Validation of Integrated Circuits Using E-beam

M.Cocito, M.Melgara - CSELT - Torino, Italy
G.Proctor - BTRL - Ipswich, UK
Y.J.Vernay - CNET - Grenoble, France
B.Courtois - IMAG/Tim3 - Grenoble, France
F.Boland - Trinity College Univ. - Dublin, EIRE

The usefulness of E-beam equipment in VLSI validation have been overstressed. However, only recently the necessity of a full integration between the CAD (Computer Aided Design) world and E-Beam world has been felt. The next step further is the complete automation of an E-beam debugging procedure. This paper presents a description of an integrated E-beam debugging system, developed under Esprit Project 271 +.

1. Introduction

The Advice project seeks to develop a methodology for automatic design error diagnosis using an electron beam. The project, partially funded by EEC under ESPRIT, sees the co-operation of three industrial partners (BTRL - UK, CNET - France, CSELT - Italy) and two Universities (IMAG - Grenoble, France, Trinity College - Dublin, Ireland). The project, started in November 1984, will last five years.

The complexity of present VLSI chips demands for powerful tools to detect possible design errors. This problem is made more difficult by two additional points:

- the information available at the external pins is a small percentage of the ones manipulated in the chip;
- the physical dimension of the internal lines makes impossible a mechanical probing without both modifying the capacitance level and destroy the interconnections.

The adoption of an E-beam allows to overcome the aforementioned problems, increasing enormously the internal observability. However, other problems arise, related to difficulties in using the instrument, the time required to position the beam, to acquire the measure, to define the debugging strategy to minimize the number of test patterns to be generated, the number of measures, the techniques to isolate the design problems.

The Advice project aims to provide the design/test engeneer with an interactive environment, integrated with the design environment, to carry out all debugging procedure in a computer assisted/aided way.

A key point of the project is the integration between design and test environment:

+ The research has been carried out within the Esprit Project 271 - Advice, Automatic Design Validation of Integrated Circuit using E-beam, partially found by the E.E.C, being contractors CSELT, Italy, BTRL, UK, CNET and IMAG, France, Trinity College, EIRE.

design data are used to make the debugging process as automatic as possible (identification of physical co-ordinates on the chip starting both from layout information and line names used in high level description, layout pattern recognition to perform accurate positioning, comparison of physical measures against simulation results...).

To achieve this goal the project intend to develop a user friendly working environment which puts under the designer finger tips all information related to the device under debug (layout information, netlists, simulation/measure results, fault dictionary and so on).

The ADVICE project was divided into two distinct but subsequent research periods.

The first one, the first couple of years, was mainly devoted to the assessment of the equipment together with all basical software tools to achieve a sort of partial automation of the operating procedures.

The second period, starting from the third year, and lasting tree years, was intended to develop a more comprehensive automatic ADVICE system, deeply integrated with the CAD environment, taking advantages of the already obtained results.

2. Description of the ADVICE System

The Advice project aims to provide the design/test engineer with an interactive environment, integrated with the design environment, to carry out all debugging procedure in an computer assisted/aided way [1].

The Advice system is organised as a multi-window, multi-menu driven program (running on DEC VaxStationII/GPX), that will provide the user with an interactive, graphical environment. The information displayed are:

- layout window;
- SEM control windows;
- waveform display window;
- text, command window.

The menus control each window and provide other commands useful for system operations (test pattern editing, background simulation, fault dictionary processing,...).

One of the major problems encountered during the debugging session of an integrated circuit, is to exactly locate the beam on the appropriate tracks: users must look for the points to be tested on the plot of the IC, then find again them on the SEM screen by moving the DUT. If we consider that a debugging session can involve a lot of points to be measured and that several times the obtained SEM images are of rather poor quality (e.g. in stroboscopic mode) it comes out that this way of working (completely manual) is hardly feasible.

By using CAD information the positioning can be moved to an assisted three steps procedure:

1) Identification of the points of interest on the layout representation;

2) Positioning using stepper motors (coarse placement);

3) Checking the correctness of the positioning and adjustment, if necessary (fine placement).

The first two steps can be easily obtained by extraction of coordinates from a layout representation (e.g. CIF format) and conversion into proper signals for stepper motors (figure 1).

The third step is a typical problem of pattern recognition and can be obtained in two different ways:

- manually by the operator, which compares the SEM chip image with the reference picture on a graphic terminal;

- more powerfully, by an automatic pattern recognition tool [3][7].

In the ADVICE system the coarse placement is achieved by using a module which allows to explore, starting from the CAD data, a layout, to extract the coordinates of the measuring points and use them to drive the stepper motors.

The coarse positioning obtained is affected by an average error of about $\pm$ 3 μm.

Once the point has been located, measurement can be performed by using a signal averager [4], and the obtained waveform is transferred to the Vaxstation.

To help the debugger to manage the results obtained, the system offers several features:

- Display and manipulation (e.g. time interval measurement) of the waveforms on a window;

- Waveform filtering and thresholding in order to reduce the noise and to obtain logic levels;

- Comparison facilities between acquired signals and logic/analogic simulations.

In order to concentrate all the operations concerning the SEM on a unique environment (again, the Vaxstation), some modules have been developed [8] which allow to control parameters like magnification, astigmatism, scan rotation and focus adjustments.

A text command window is present which can be used as a communication way between the ADVICE system and the rest of the world, intended as the operating system of the Vaxstation.

By integrating the previous described modules, the flow of operations during an ADVICE debug session is reported in figure 2. This flow can be interpreted either as assisted or automatic procedure by only substituting the "Determine next node" box with the probing algorithm described below.

In summary, first research period results have been:

- Computer control of some EBT parameter;

- Automatic placement of the beam, starting from CAD data;

- Computer control of waveform acquisition;

- Development of hardware and software procedures for image acquisition, handling and recognition;

- Development of hardware prototype and software procedures for fast waveform acquisition and averaging;
- Practical evaluation of the E-beam capability in controlling chip internal nodes;
- Theoretical researchs on waveform sampling techniques.

All the aforementioned results represent a thick background on which to begin the growth of the final ADVICE system. The present system has allowed to cut the time required to acquire 15 waveform from about 2 hours, at the beginning of the project, to few minutes (figure 3).

3. The final Advice system

The second research period, as mentioned before, aims to integrate all previous results into a unified debugging system, increasing the part related to error detection methodologies.

Although many technical problems, still related to the E-beam equipment, need to be solved during the next research years, the basis of the final system can be sketched.

The leading lines for the system definition will be:

1) The ADVICE system will be a user-friendly environment in which a designer will perform all operation related to the debug procedures, without leaving it, but having under his finger-tips all tools needed.

2) The ADVICE system will be linked to the existing world through standard interfaces (languages and procedures), that will allow the development of a single nucleus, in co-operation by all the partners.

3) The ADVICE system will run on a common workstation (Vax Station), eventually integrated with dedicated hardware.

4) The ADVICE system will automatically perform all trivial, repetitive and tedious operations, leaving the designer free of concentrating on decision tasks, eventually suggesting him basical strategies.

5) The fully automation of the debug procedures could only be achieved, with the present technology, for combinational circuits.

To achieve those goals three main tasks have been defined:

- Development of image recognition techniques;
- CAD development and integration;
- Strategies for debugging.

The first one will continue the work started in the previous period, coming to a final product composed by software, and eventually hardware, tools to perform a fast fine adjustment of the beam position, after the rough positioning obtained using stepper motors, driven by layout information.

The second task, CAD integration, will produce the user-friendly environment, by implementing the connection towards the design and simulation data, and the physical equipments (E-beam microscope, circuit exercisers...), and developing the user-friendly working environment.

The task on strategies will face essentially four problems [6]:

- design for electron beam debugging: it will provide design rules and probing point selection rules to make a debugging session satisfactory;
- simulation strategies: it will provide guidelines to the IC designer about the techniques to be followed during the circuit simulation and test sequence generation;
- fault dictionary generation and manipulation: some knowledge about the modes of failure can be pre-computed, stored in a fault dictionary and used as starting point for the debugging procedure, to cut the number of measures to be performed;
- development of the diagnostic tasks.

The diagnostic tasks, seeing the Advice interactive system as procedures able to interface them to the physical reality, aim to perform a fully automatic debugging session.

Starting from the knowledge about the logical model and the layout of the circuit, the simulation results and the status of the debugging procedures, the diagnostic tasks select next points to be probed, compare the measured and the simulated results, trying to locate the failing IC area.

4. Conclusions

The Advice project, which has reached the 2/5th of its duration, has obtained till now very good results both on the technical and the international co-operation bases.

As a practical result it can be said that the time required to perform a measurement with the E-beam equipment has been reduced, respect to two years ago, to about one sixth of previous one.

However, a long way is still to be climbed during the last three year period to achieve the goal of a fully integrated debugging station. (figure 4).

Acknowledgements

The authors are pleased to acknowledge F. Stentiford, J. Dowe, T. Twell (BTRL), M. Battu', P. Garino (CSELT), I. Guiguet, D. Micollet, M. Marzouki (IMAG), E. Linch, K. Hundertpfund for their indispensable contributes to the research development. We also intend to thank D. Ranasinghe and D. Machin (BTRL), G. Bestente (CSELT) for their useful work in ameliorating the E-beam system.

Bibliography

[1] M.Cocito, G.Proctor, Y.J.Vernay, B.Courtois, F.Boland: "Advice: A European effort", 1st Europ. Conf. on Elect and Opt. beam testing of IC, 1987, to be published.

[2] M.G.Battu', G.A.Bestente, P.G.Cremonese, A.B.Di Janni, P.A.Garino: "Automatic positioning for electron beam probing", XI Int. Congr. on Electron Microscopy, 1986, pp. 651-652.

[3] F.Stentiford, T.Twell: "Automatic registration of scanning electron microscope images", 1st Europ. Conf. on Elect. and Opt. Beam Testing of IC, 1987, to be published.

[4] D.Machin, D.Ranasinghe, G.Proctor: "A high speed signal averager for electron beam test systems", 1st Europ. Conf. on Elect. and Opt. Beam Testing of IC, 1987, to be published.

[5] E.R.Linch, F.Boland: "Parameter extraction in E-beam testing", 1st Europ. Conf. on Elect. and Opt. Beam Testing of IC, 1987, to be published.

[6] M.Melgara, M.Battu', P.Garino, J.Dowe, M.Marzouki: "Fully automatic VLSI diagnosis in CAD-linked E-beam probing system", 1st Europ. Conf. on Elect. and Opt. Beam Testing of IC, 1987, to be published.

[7] I.Guiguet, D.Micollet, J.Laurent, B.Courtois: "Electron beam observability and controllability for the debugging of integrated circuits", XII Europ. Solid State Circuit Conf. 1986, pp. 181-183.

[8] D.W.Ranasinghe, G.Proctor, M.Cocito, G.Bestente: "Computer control of Electron beam testing for design validation of VLSI circuits", XI Int. Congr. on Electron Microscopy, 1986, pp. 619-620.

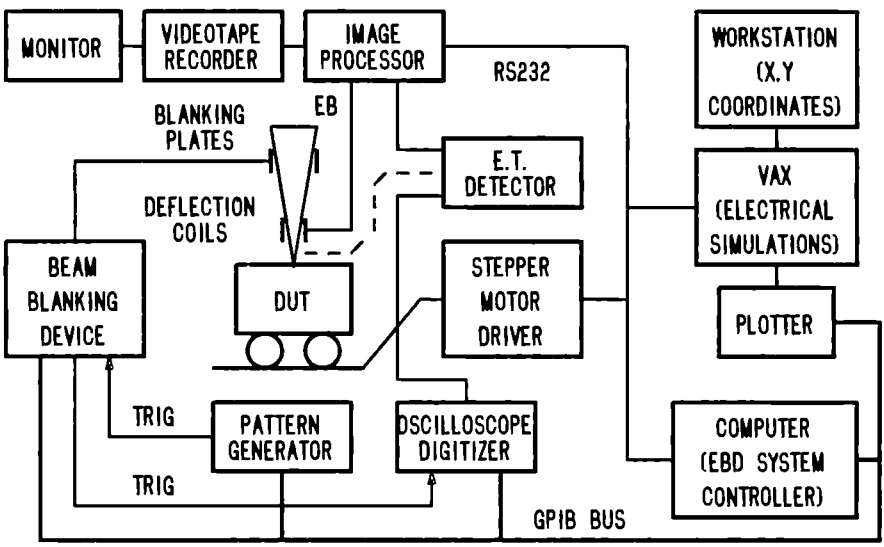


Figure 1: Computer controlled electron beam debugging system.

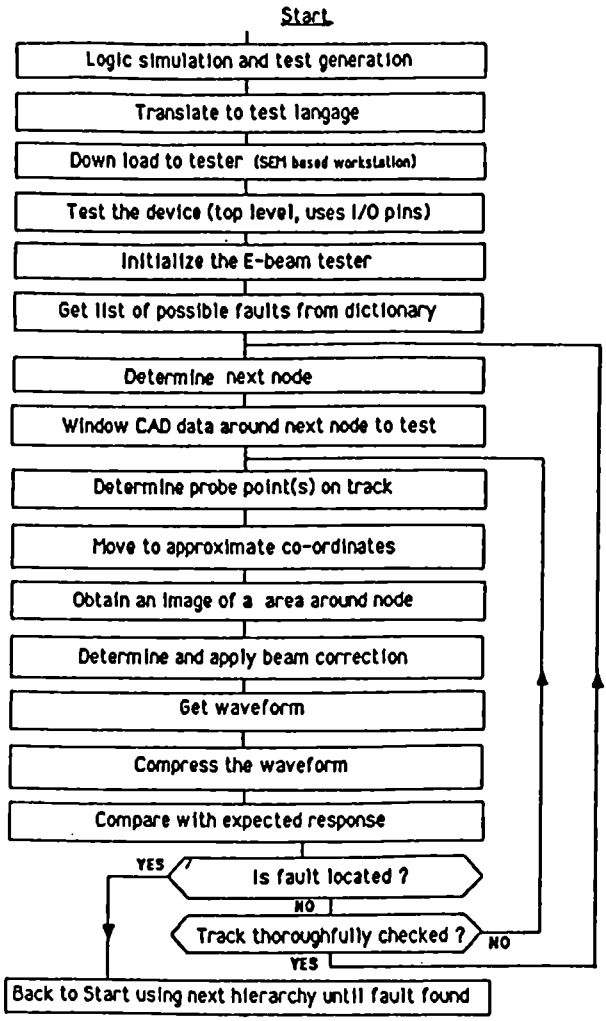


Figure 2: Advice system operation flowchart.

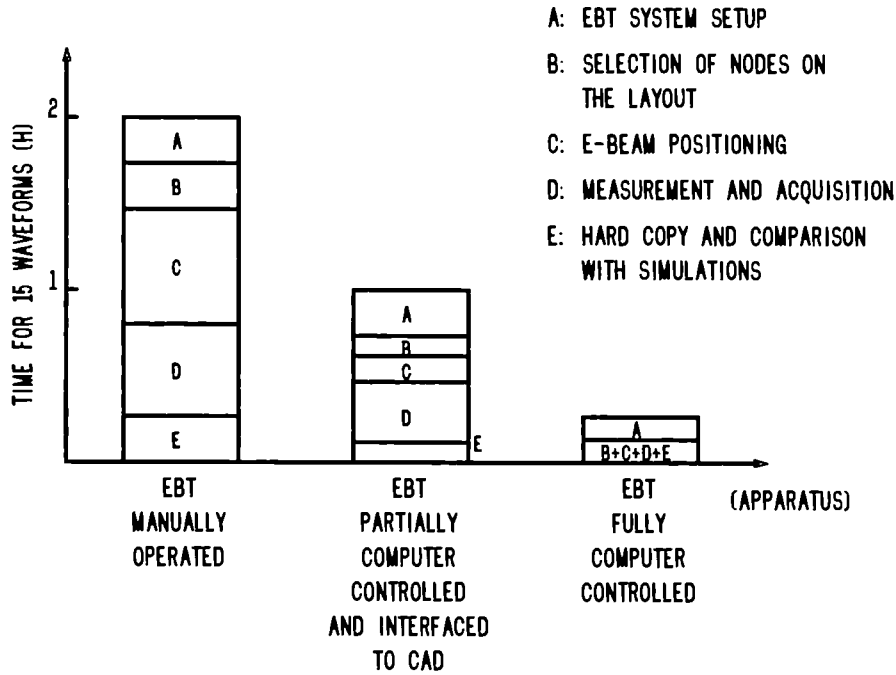


Figure 3: Electron beam debugging system throughput.

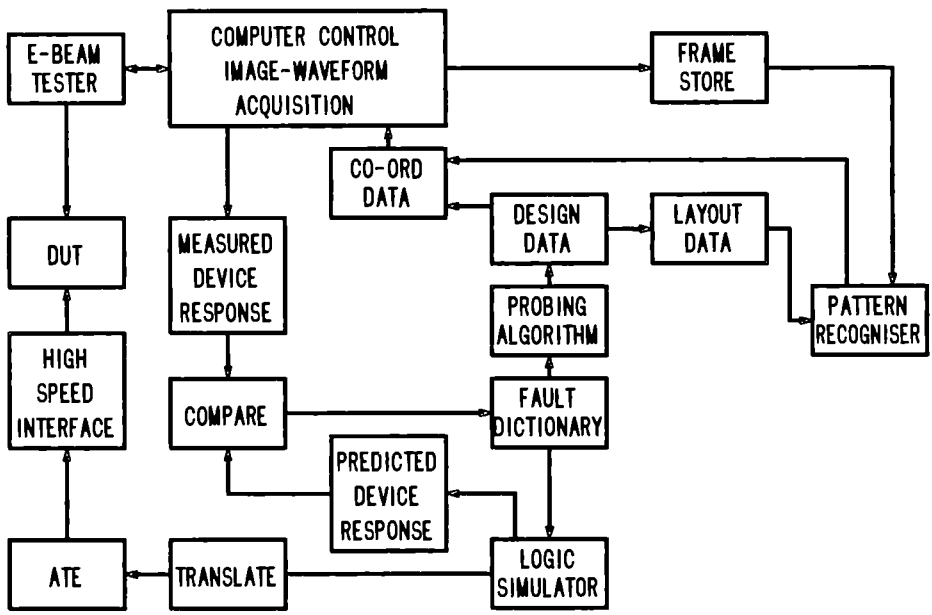


Figure 4: The final Advice system.

Project No. 888

AIDA

ADVANCED INTEGRATED CIRCUITS DESIGN AIDS

AIMS AND PROGRESS TOWARDS A NEW GENERATION OF VLSI TOOLS

H.G.THONEMANN, AIDA Project Leader
SIEMENS AG, CORPORATE APPLIED COMPUTER SCIENCE LABORATORY
MUNICH, WEST GERMANY

ABSTRACT

Aims and progress of the AIDA project are presented. AIDA is a Research and Development project for new VLSI design methods and corresponding CAD tools to master the growing complexity of VLSI circuits. The three main partners in the project: ICL, THOMSON and SIEMENS in cooperation with the subcontractors BULL, IMAG and UMIST are working on a new generation of tools that will help the designer to apply his creativity for efficient and optimal design solutions. The program began in late 1985 and has completed the first year objectives. In particular the focus is on object-oriented data management of VLSI data, capture mechanisms of system specifications, logic synthesis, floorplanning and advanced placement and routing, a new generation of test aids and user interface issues. The team members are presently involved in the specification phase for concepts and tools following the workplan.

INTRODUCTION

The Aida Project is a Reseach and Development project for new VLSI design methods and corresponding CAD tools mastering the growing complexity of VLSI circuits.

The three main partners in the project ICL, THOMSON and SIEMENS in cooperation with the subcontractors BULL, IMAG and UMIST are working on a new generation of tools that will help the designer apply his creativity for efficient and optimal design solutions.

The quest for appropriate chip architectures, decomposition into the correct blocklevel and logic design, application of optimizing algorithms for layout and consistent representations across all levels of design abstraction was the basis to divide the AIDA program into seven work packages:

1. Data management and design control
2. High level specification capture
3. Logical and electrical synthesis
4. Physical Design and Layout
5. Testing and testability rules
6. Man-machine interface
7. Evaluation

A fundamental objective of the four-year project is to develop concepts, methods and experimental tools in these working tasks and integrate the tools into the existing CAD environments of the partners. An important step applies the experimental results for the development of industrial production tools.

Major axes of the work packages include the application of artificial intelligence techniques on siutable problem areas such as logic synthesis, testing and floorplaning, the

improvement of data management techniques and design control, and some improvement of traditional tool sets.

The objectives of the first year of the program were completed in late 1986 and delivered to the partners and the EEC during the 1st quarter 1987. During the first half of 1987 a number of tasks have produced further results. This presentation intends to publicize the present status of the project..

DATA MANAGEMENT

At the commencement of the AIDA contract it was recognized by the three partners that an integrated object-oriented database management system is of prime importance for tool integration and future evolution of the design environments. Thus one of the initial main goals of the task was to work towards the formulation of requirements, specifications and implementations of some first facilities for advanced data handling, design control and design release.This work forms a basis to support effectively, designs of increasing complexity with short design time requirements. The work todate has covered activities on the following issues:

- Specification of the object-centered approach
- Formulation of requirements and specification of conceptual data models
- The formulation of a common, consistent conceptual data models notation
- Language specifications for data definition and data manipulation
- Concepts for portable interfaces
- Control of design completeness and consistency
- Design process control
- User interface to design control facilities
- Management and control of external objects
- Methods for data distribution
- Formulation of a technology specification language
- Support for design audits and design release
- First implementation of object-centered prototype components

The specification of the object-centered approach describes a set of concepts for organizing common design data by using abstraction mechanisms drawn from Artificial Intelligence (AI) techniques. This approach is well suited to the consistency control required of an integrated CAD environment.

One of the partners is presently completing a first prototype of an object base management system (OBMS) using a LISP environment. It implements the basic concepts of the object-centred approach and permits the storage of IC design data plus the control of object base semantic integrity.

Transparent distribution of data was evaluated and a set of requirements formulated. Experimentation will be performed on SUN-Workstations to formulate optimal node/server configurations and determine an optimized store size per node. Evaluation of communication between workstations and host will be based on ETHERNET

A control environment has been implemented at one of the partner's system which will support the creation, extraction and manipulation of data in external objects. Further evaluation of this support infrastructure will show the performance of controlling a set of data bridges to external environments, the handling of any new external interfaces via external objects and the conversion of existing interfaces.

A first functional specification defining the syntax and constructs for a technology specification language has been formulated. A technology compiler and run-time support infrastructure are being implemented. The use of technology objects to specify base

technologies, enhancing technologies and to meet user specific requirements will require a high level of support

Since there is no declared intention for SIEMENS, THOMSON and ICL to develop or adopt a single data management system as part of the AIDA project, the collaboration within the other task groups in AIDA through the exchange of data requires data management support. As a first attempt for such activities a comparison of the netlist *conceptual data models* for the three systems was performed; the presentation format was *Bachman diagrams*. This work exposed severe limitations with Bachman diagrams and resulted in the proposal of an alternative *object notation* by SIEMENS. This notation had the capability to accurately represent complex design object structures and relations.Requests of the partners for extensions to the basic notation for the handling of generic objects and changes to simplify the handling of relationships were included. As a result, a common AIDA notation for the presentation of conceptual data models was formulated and agreed. A full definition of the standard is presented at this conference. The production of this standard represents a significant achievement and a solid basis for further collaboration

SYSTEM SPECIFICATION

The work up to the present state has involved the production of a requirements specification for capture and simulation tools. The investigation into the requirements for system specification has made it clear that a single Hardware Description Language does not provide the best means of defining a complex system design. For this reason this objective of the project has been redefined as 'Develop an environment for the capture and verification of a complex VLSI system'. In addition performance and user interface are seen as top priorities for the simulation tools. Thus the major activities have been concentrated on:

- Production of a detailed requirements specification for the simulation system.
- Production of a formal definition of aggregates and extensions to the capture system and simulator to fully support aggregate types.
- Investigation into new scheduling algorithms for system level simulation.
- Provision of a mechanism for the capture of state.
- Implementation of a first set of prototype facilities.

The requirements specification for system specification has been completed. The production of a formal definition of *Aggregates* is complete and includes a BNF definition of the aggregate syntax, definition of aggregate access formats and notes on building aggregates graphically. During the work the importance of state or memory was recognized. The capture facilities are therefore being extended to provide a means of specifying a 'unit-of-design's' memory in terms of a simple size (no of words or no of bytes) or as an aggregate form. This information can be used by the simulator and the 'unit-of-design's' behaviour.

Work on the simulation toolset has identified a number of areas for potential performance improvements such as critical inputs, multibit events, timeloop sizes, 2-state behaviour and transparency.

LOGIC SYNTHESIS

Synthesis is the process of translating a description into a lower level equivalent. The objective of the logic synthesis in AIDA is to cover, step by step, the full range of synthesis problems - from high level, behavioural descriptions, to detailed logical and electrical schematics. Currently, the conversion of the final schematics into layout is not an integral part of the task.

The highest description level addressed is associated with a *system source language*, BEADLE (BEhAvioural Description Language Environment), which provides for architecture independent, algorithmic descriptions. Using this, systems consisting of one or more IC's can be described in terms of communicating sequential processes.

Another aspect of the task concentrates on logic synthesis techniques that begin with architecture dependent, algorithmic descriptions. These are stated using a *Concurrent Algorithmic Programming language*, which is also suitable as a modelling language for multi-level simulation.

This high level approach is therefore complementary to the BEADLE equivalent. Both descriptions are translated into a common intermediate form, *Flowgraph*, in which systems are described using control and data graphs. Then the synthesis of a chip proceed from a Flowgraph description. An intermediate abstract structure format that is independent of cell libraries has been devised from which the expansion into modules of a particular target library can be performed. It has been agreed to use the *Flowgraph notation* as a common high level notation between the tools of the three partners. The abstract structure format serves as the link to specific low level Technology-near implementations.

A third package of the work of this task has been essentially devoted to basic synthesis and extraction mechanisms, using AI techniques. A unique formalism has been defined to deal with the problem of cell synthesis, down to the switch level. Experimental tools handling combinatorial CMOS cells, have been realized, and are currently being experimented with. The philosophy is to embed the design style within sets of rules, allowing a final realization in terms of either predefined gates or predefined "topological targets". The latter allow the production of a complete layout, but are independent of existing cell libraries - although dependent upon the capabilities of particular technological families.

The work has been concentrating up to now on the following topics:

- Mapping of the BEADLE behavioural descriptions onto a target architecture
- Implemention of interprocess communications by customising a generic architecture
- Production of an initial version of the BEADLE language
- Development of a generic architecture for Behaviour-Architecture Synthesis
- Investigation of low level physical architectures (for example busses)
- Prototype implementations of the BEADLE Direct Input System and Database
- Investigation on simulation as a part of tool integration
- Compilation into a Flowgraph representation (combinatorial logic)
- Multi-level synthesis using primitive cells
- Definition of an abstract structure format (independent of cell libraries)
- Abstract structure expansion into modules of a particular target library.
- Synthesis of controllers with state diagram inputs and realization with FSM

PHYSICAL DESIGN AIDS

At the start of this project powerful systems for layout generation were in place at a large number of organisations dealing with the problem. This was especially true for standard cell design where special placement and routing algorithms allow for reasonably automatic layout generation. In some instances the physical design with macro cells had already been reported. This design style however seemed not yet put widely into practical use. Furthermore an entirely consistent top-down approach to placement and routing could not be observed. Therefore new approaches needed to be taken to tackle the hierarchical generation and verification of layout geometry data. The task concentrates presently on four major issues:

Floorplanning

The exhibition of the placement problem to hierarchy becomes a floorplanning task dealing with blocks whose geometric characteristics are not yet fixed and taking routing spce into account for interconnects, whose terminal positions are not yet known either.

A good portion of the recent work has therefore been devoted to study a number of approaches for the solution of this problem:

- Effects of logically equivalent cell terminals on the reduction of routing space
- Dynamic effects on power & ground routing for the minimization of routing area
- Investigation of simulated annealing techniques for interactive placement
- Empirical evaluation of sizing models
- Evaluation of graphical editing applications to network and layout views.
- Determination of effects from the merge of autoplacement tools with interactive placement approaches
- Formulation of requirements based on the evaluation of available floorplanning methods and tools
- Integration of available floorplanning facilities and experimentation to identify further bottlenecks.
- Placement analysis to define algorithmic strategies versus expert system approches
- Specification of expert system aspects: LISP inference engine, circuit typology and rule collection.

Layout Algorithms

In connection with the floorplanning work a number of routing algorithms have been studied by a few members of this task:

- Experimentation on automatic two step layout generation to a given target structure from an input net list. (The target structure chosen for experimentation is made up or two rows of cells separated by a channel): arrangement of the cells in the rows and routing of the channel.
- Automatic compaction or expansion of a layout.
- Experimentation on compaction and routing sequences

Cell Composition Tools

Earlier investigations indicated that symbolic layout tools ("stix" editor and compactors) provided the simplest path and required least effort to maintain process independence. However such systems are intrinsically inefficient due to the fixed topology of layouts. Therefore the work has concentrated on the following:

- Generation of process efficient topologies for cells from circuit diagrams
- Formulation of the requirements for cell shape generation tools using size, shape and power consumption as parameters.
- Implementation of Prolog prototypes for cell implemetation on a simple double strip transistor arrangement
- Modification of process rules to determine the effect on the cell layouts

Hierarchical Design Rule Check

Hierarchical layout verification means the recursive check of all cells of a given layout. Each cell has to be checked completely before its instances on higher level can be checked against their environment. This implies repeated access to the layout data. Therefore the geometry database needs to be structured for this purpose something that is not sufficient in most systems today. The main results today are:

- Requirements for hierarchical design rule check based on applying an existing DRC-program recursively to the cells of modified layout data
- Evaluation of existing solutions for hierarchical design rule checks with ECAD DRC DRACULA III to be the most interesting approach because of hierarchical error report.
- Redesign specification of existing DRC programs: Unification of Boolean operations with design rule checks as a basis for hierarchical design rule check
- Specification of interactive capabilities for verification (DRC) and assistance for parameter estimation of topology, performance and testability

TESTING

As the complexity of chip continues to grow the testing problems follow suit. The AIDA objective is to develop methods, algorithms, languages and tools for *Design For Testability, Self-test* and *Test Pattern Generation* to overcome deficiencies of classical approaches and assure design quality in the future. Extensive results have been achieved since the start of the project and are presented in condensed form

Design ForTestability

- Preparation of a survey of DFT guidelines & rules of each partner, distribution and discussion of commonalities and divergencies for further definition of common approaches.
- Testing of Embedded Regular Structures (e.g. RAM) with determination of area overhead, fault coverage and test time. Completion of logic structures for RAM BIST.
- Study on design for testability of PLAs, especially of optimized PLAs.
- Study on design for testability of control sections and datapaths
- Feasibility study on expert analysis tools for testing style advice and minimum logic for test.
- Extension of a DFT rule checker to check for the correct applications

Test Method Language

- Completion of a formal definition of a Test Method Language with examples of its use for defining test methods.
- Implementation and release of a complete VLSI CMOS cell library of Test Methods. Development of Enhancements
- Investigation of valuable feedback from designers on potential extensions to both language and tools.
- Development of a test program integration aid tool with definition and compilation of test procedures expressed in TGL language and generation of the test program from device description, test flow, requested procedures and truth tables
- Specifications of a test description language UTILE (Unified Test integrated Language) for unification of various kinds of simulators and different types of ATE. Syntax structures for description of test patterns, input stimuli and expected responses as well as timing data, tester informations, simulation control and documentation.

Built-in-Self-Test

- Extension of a bit- pattern- generator for generation of initializing, controlling and evaluating patterns of self-test structures.
- Investigation into the requirements of an on-chip self -test diagnostic unit.
- Study on Unified Built-In Self-Test (UBIST) technique for testable PLA's using self-checking and built-in self-test (BIST) techniques to ensure on-line (or concurrent error detection) and off-line testing
- Design of Self-Checking and BIST control sections with folded and/or unfolded PLAs implementing on-line and off-line test features for stack-organized control sections (slice architecture)
  - ★ On-line test by means of Self-Checking schemes (i. e. concurrent detection of errors) using Strongly Fault Secure functional blocks and Strongly Code Disjoint checkers.
  - ★ Off-line test with a BIST scheme for internal test pattern generation and verification
- Study on self-test methods for RAMs with emphasis on parametrizable RAMs.
- Study of direct access test methods for RAMS and ROMS and their support by the test equipment.

Fault Simulation

- Development of extensions for an existing logic simulator towards analog modeling and exploitation in order to describe analog functions by their transfer function, expressed in synchronous linear equations with logic input and output signals
- Performance improvements of the same simulator for fault simulation mode using a concurrent algorithm

Analog Testing

- Development of a general concept for automatic test hardware synthesis based on standardized elementary test circuitries and configurable device specific test set up. Initial design and implementation of the synthesizer algorithm. Study on combinatorial optimization techniques,especially on 0-1 linear programming problems and related topics, such as LP relaxation and cutting plane methods.
- Definition of an ATE software architecture: determination of target devices to be served by automatic analog / digital test program generation and specification of a test parameter set for analog standard cells.
- Specification of the requirement for an automatic test program generator system library and definition of the description language for the electrical parameter set, test system configuration and test set up configuration. Determination of the library model

Automatic Test Pattern Generation I Test Quality Verification

- Extension of ASTA for generalised designs with emphasis towards automatic test pattern generation.
- Study on Fault models and algorithms for test pattern generation for non-structured combinatorial circuits.
- Study on Optimized Testable Programmable Logic Arrays. Review of Optimization Methods and Off-Line Test of PLAs with the presentation of problems and solutions for folded PLA's with built-in testing
- Optimization and self-checking design of PLAs based on the use of parity codes and the use of Berger codes
- Improvement of the D-Algorithm: Reduction of Backtracking and Study on D-Algorithm Alternatives
- Simulation under Consideration of Automatic Test Equipment Constraints (Dynamic Testing)
- Analog Test Program Generation: Method definition and specification of a parser
- Test pattern generation at the switch level with node analysis of given a switch network looking at neighbourhood and influenced area of a node in order to determine path sensitization is chosen as a useful approach.
- Definition of off-line tests through deterministic test pattern generation and signature analysis using ***L***inear ***F***eedback ***S***hift ***R***egisters

MAN-MACHINE INTERFACE

The potential benefits of a new generation of design tools will only be ralized if users are provided with comfortable facilities that support their needs in an efficient and consistent way. Therefore the project activities concentrated on the following topics:

- *Workstation Management* concerned with the definition of a general software architecture and the specification of the services to be provided by the operating system in order to optimize the Man/Machine interface. Todate an analysis of X-Windows (MIT Vers. 10) for protocol implementations has been completed identifying the need for
  - ★ a multiple-wait facility that is known as the "select" system call in UNIX BSD
  - ★ mechanisms known in UNIX BSD as "pseudo-tty"
  - ★ Integration mechanisms, preferably UNIX BSD "sockets"for protocol calls.

- *Display Server* concerned with the definition of a display formalism, and the implementation of a display interface. Results:
  - ★ Study of the available specifications of X-Windows version 11 and experimentation on a few programs with version 10 available on SUN-Workstations
  - ★ Definition of necessary targets for porting the X-Server:
    - □ the operating system dependent routines must beported.
    - □ the protocol routines must be compiled, and this might imply some tuning of individual C compilers.

  - □ a number of device dependent routines must be implemented (one of them being the routine for printing characters on the screen). On X-Window tapes, all device dependent routines are provided for VAX- stations. One might try to adapt them to specific hardware or implement them differently.
  - ★ Study on the portability of fonts and tuning of fonts for different hardware

- *Man-Machine Dialogue Tool Kit*
  - ★ Study of the X-Windows *WIDGET* (Window Object) concept with object oriented flavours
    - □ Object reaction on input actions
    - □ Replacement of default widgets by complex applications
    - □ Preparations for AIDA applications
  - ★ Experimentation for effects of X-Windows programming style on I/O management of applications

EVALUATION

Presently the project is going through the specification phase for the tool sets of the respective partners. Specification results are the main objective for this second year of the project. The verification of choosen approaches, however, is essential for the completion of major project milestones. For this reason the evaluation task was created. It started to involve efforts for the definition and preparation of validation test sets. Discussion among the partners was initiated in order to assure commonality during update phases.

CONCLUSION

Results of the AIDA project have been presented. After a year and a half of work the AIDA project has produced a number of solid building blocks for a new generation of VLSI CAD tools and provided its partners with advanced methods for the integration of these tools into their cohesive design environments. The pursuit of collaboration during this period has served as an important cornerstone for discussing and clarifying the respective objectives of each task and to understand the background on which the actual work at each company is going on. This will be essential when the AIDA project is moving into the prototype implementation phase within 12-months time.

ACKNOWLEDGEMENTS

The author would like to extend thanks to all team members of the AIDA project for their support and expert contributions to the VLSI design automation tools presented in this paper. He is in particular grateful to Mrs. B. Martin who has integrated the multitude of textual contributions into a single presentable format.

Project No. 802

CAD OF ANALOG CELLS

Günter Wagner*
GMD, St. Augustin, FRG

1 Introduction

Whereas large effort up to now has been spent in design aids for digital integrated circuits, a corresponding development in analog integrated CAD has not been occurred.
Digital circuits mainly distinguish only between three logical levels (High, Low and High impedance) and are generally controlled by a clock. This behavior makes them very useful for application in computing machinary.

Analog circuits instead have to process all values continuously within a defined voltage or current range. The structures in general are not very regular; influence of effects like nonlinearity, distortion, noise etc. plays an important role and must be considered very carefully. Appropriate to the increasing demands on speed, reliability, costs and lifetime of components in control mechanisms, sensor or switching circuits, analog integrated circuits get a very fast increasing application area.
But, unfortunately there are no comfortable design tools to support the development of this kind of ICs.
In CVS, work package 3 we aim at closing this gap and propose a system to give the designer a powerful tool to support the design of analog integrated circuits, if wanted from scratch.
The system is to lead the designer through the whole design process and will be highly interactive. Of course, if an already designed cell meets the specification of the needed circuit, this cell may be used unchanged for a possible implemention within a circuit of higher complexity. If, however no cell of the library fullfills the demands of the specification at least for part, a complete new design starting from scratch must take place. The most convenient case, however will be the third one:

*G. Wagner is project manager of the EEC CVS project (No. 802) work package 3. He is with Gesellschaft fuer Mathematik und Datenverarbeitung mbH, Grossprojekt E.I.S., P.O. Box 1240, 5205 St. Augustin, FRG

The cell demanded for meets the specifications for part, that means, the user will be able to modify or to supplement an already existing cell of the library to meet the specifications.
The designed circuit will be checked only for functionality by using a circuit resp. a SC simulator like SPICE, BONSAI or DOSCA. An additional check on layout errors (DRC,ERC) will not be performed. By using the system, where layout is correct by construction, these checks are superfluous.
This way guarantees both a high reliable and fast development cycle as well. It demands for a high sophisticated design system, able to guide the user through the design process, giving proposals or hints how to proceed during the design of a special analog circuit.
The system uses AI techniques and object oriented programming. Knowledge based synthesis is provided in the schematic (topology and parameterization) as well as in the layout design phase.
The system may be adapted to any technology by providing the necessary rule base. For that purpose, a large number of small hierarchical rule bases instead of a large one is involved.
The paper describes in general terms both, the concept of so-called skeleton cells as well as the complete design cycle. The system allows expansion both in rules and scope. The input of rules can be done either in text form or graphically.
In the 1. subsection the User Interface (UI) will be presented. The 2. subsection explaines the concept of so-called 'skeleton cells' their use and the information available in the library. Finally in the 3. subsection, the implemented simulators with their extensions and optimizer are presented. The conclusion will show up the results reached up to now and the way how to proceed within this project.

2 The system

2.1 User Interface (UI)

UI stands for the interface between user and system. It has to manage and to control the design process (see fig. 1).

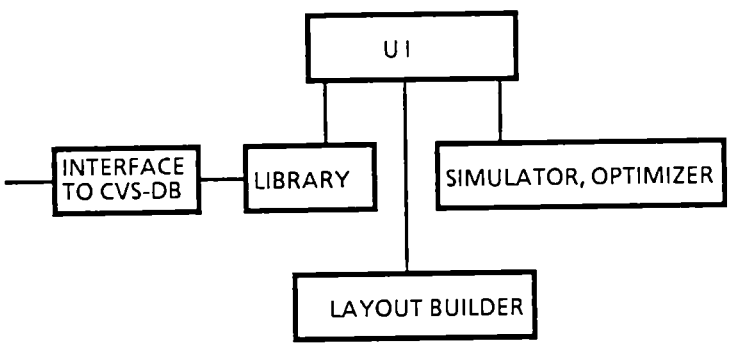


FIG. 1 THE SYSTEM ARCHITECTURE

2.1.1 Architecture

The UI is menu driven. That means, that for all activities within the system a special menu is available to support the user during the design process. Using 'icon's makes handling of the system very easy and comfortable (see fig. 2).

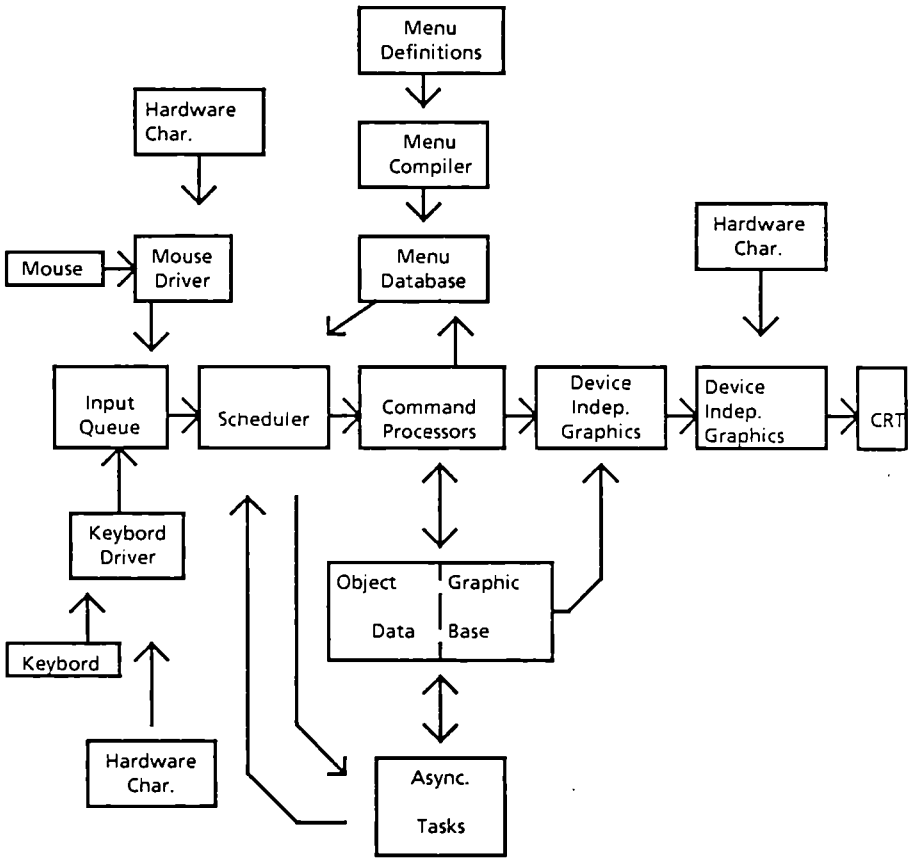


FIG. 2 Structure of user interface

For the different steps of the design (schematic, layout) , editors of different types are

available. To input textual information like rules or simulator input files, a suitable editor has been generated. Secondly, to input or output a schematic, a schematic editor is provided. It is to give the user a graphical overview on the circuit he is working on.

Schematic will be input by clicking the corresponding components like resistors, capacitors or transistors at a menu displayed on the screen. As will be explained later, not only the graphical information of a component is available, instead there is a complete data structure appended to each symbol selected from menu. Furthermore UI has to enable the input of rules (textually or graphically) for e.g. another technology data or to add new knowledge about the design process.

Based on the specialized structure of the editors, definition of 'macro's is possible. These macros contain information about naming, the graphical representation, ports where their instances can be connected, a set of component or macro instances, junctions and a set of rules. An additional netlist (perhaps extended) will be automatically generated during the schematic input process. This netlist, possibly together with some additional information of the circuit like 'nodesets' or similar parameters represents the data part of the simulator input file. Of course, simulators will be started by the UI.

Another task of UI is to support the interactive generation of the layout. So, it must be able to display the layout, generated so far and additionally allow to perform modifications on it.

All tasks, described above, are managed by the UI. Each design step is supervised by the UI. It has been made to enable guidance of the user through all steps of the design process.

2.2 The concept of skeleton cells and their use

2.2.1 Cell generation

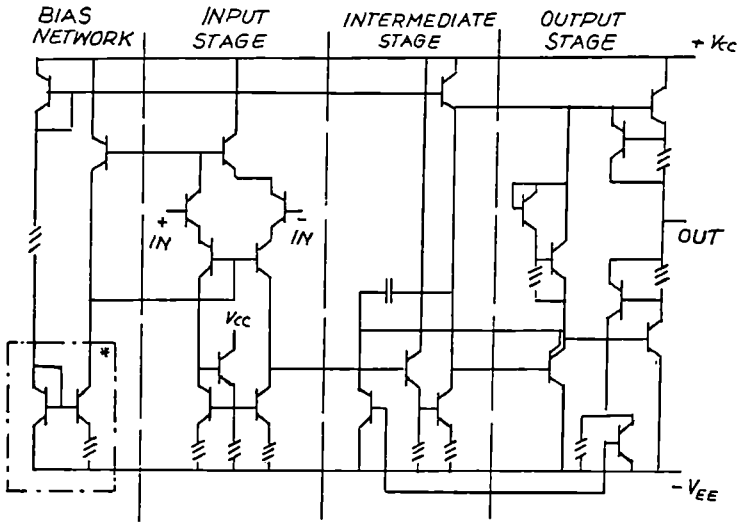
Cell generation entails the user specified refinement of an implementation. The user is presented with an implementation consisting of components and so-called skeletons connected in a particular fashion.

Given certain requirements, the user interactively specifies which subunits of the cell he wishes for further refinement. Skeleton cells within this system are 'sub-parts' out of which complex circuits are built. For demonstration let's take an operational amplifier (OPAMP, see fig. 3). This circuit can be divided into e.g. four parts, the part to generate the bias, the input differential stage, the intermediate stage and the output stage. These parts themselves consist of e.g. current mirrors, level shifters, differential pairs and so on.

They are represented by 'skeleton' cells. These are data structures containing the necessary framework used in different phases of the design activities. They are active data structures which allow to generate an OPAMP with very specialized specifications.

Dependent on the special application, these parts look quite differerent. That means, to meet for example more restrictive specifications - a cascode connected current mirror to increase the dynamic resistance of a stage - a particular 'implementation' of a skeleton (see fig. 4) will be needed. Cell generators (see fig. 5) to

generate the cell layout, are attached to the possible 'implementations' of a 'skeleton'.



* ONE POSSIBLE SKELETON CELL IMPLEMENTATION OF A CURRENT MIRROR

FIG. 3 OPAMP

If a design contains cells whose implementation is not fixed, the user designs the subunits in a top down manner to meet the overall specs. He may pick the selected cell in the editor and invoke the system (by application of rules) to generate it. The old skeleton implementation will be replaced in the cell by the new generated one. This can be done either automatically or highly interactive.
During the design shown above, the parameterization steps are to determine the topology and the parameter values of the cell.
As we have demonstrated above, the design of analog cells is not only supported by the system to link already existing analog circuits, it instead is supported in the design process some levels lower. By this kind of design a real customized analog integrated circuit generation is enabled.

2.2.2 New cell layout design

In order to produce a layout, the system must be initialized with a full set of technology files and in addition a set of component generators in minimum. In general, generators within this system are functions which produce the layout of a number of

components which together form a well defined circuit function.

The generation process can be divided into five steps, namely:

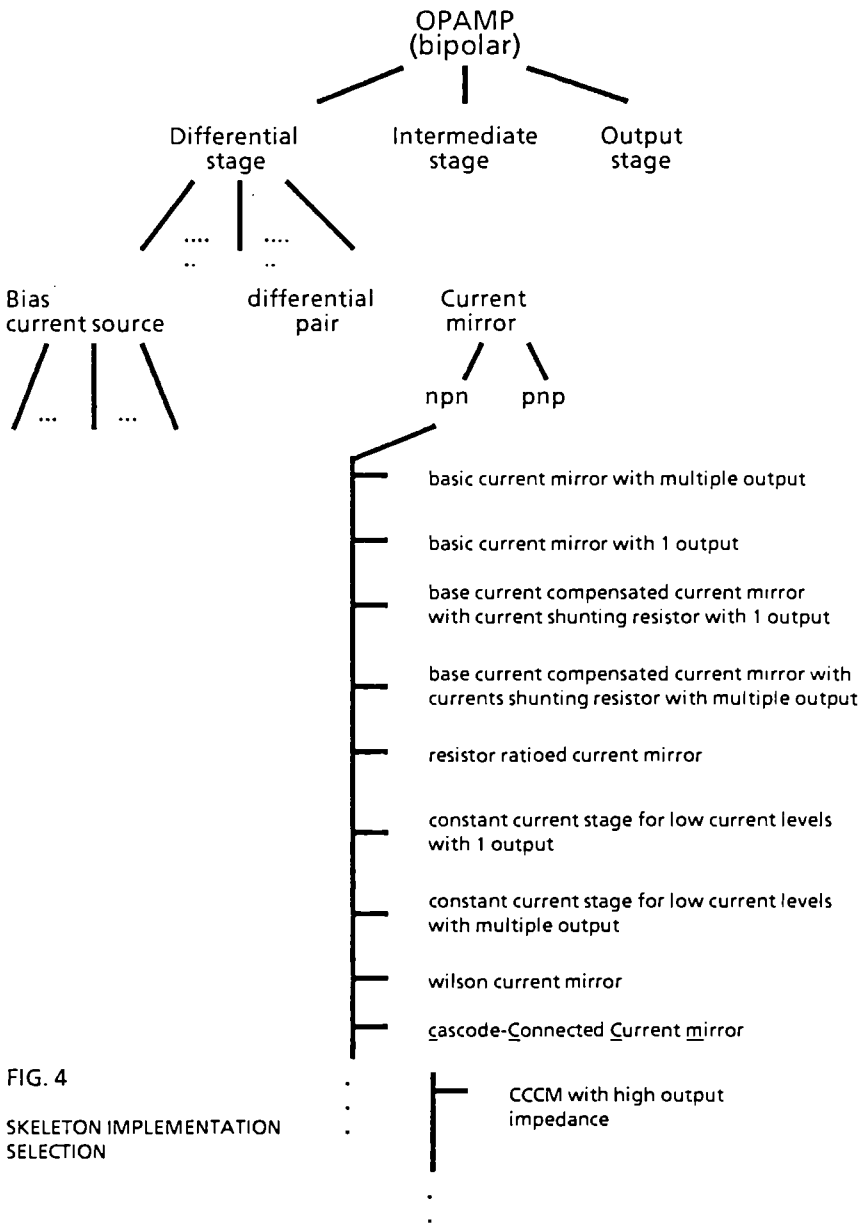


FIG. 4

SKELETON IMPLEMENTATION SELECTION

- **Definition of the outline and the position of ports**
  Prior to the actual design, the user must define the interface of this cell to its surroundings. If the user specifies nothing, the layout system will try for a square cell and place the ports without contraints. If, however the space allocated by the user is not enough, he will be notified by the problem as soon it is detected and is given a suggestion as to its correction as well.

- **Selection of component types**
  This is the process of selecting the layout implementation of components specified at the schematic level.
  In order to start the layout design phase, components must exist at the schematic level and must be imbedded in a netlist implying that all their ports are loaded. The loading values affect the layout structure. Another factor which may play a role in this selection is the way sets of components react to each other. Examples are current mirrors, loading stages etc. The generation of these structures is performed by 'cell generators'. The dimensions of the selected components (e.g. width to length ratio of a transistor) are under user control; the actual structure however is fixed.

  The selection of a generator for a circuit component or a set of them can be altered at any time. The system automatically recognizes when a selection needs to be updated, if for instance, some of the conditions responsible for previous selection have been altered. In this case, the system recognizes the changing automatically and gives a warning of the eventual incompatible layout implementation.

- **Floorplaning**
  This is the process of placing the different components of the circuit relative to each other. The information about the mask geometry of a component or how to place it on a circuit is laid down in 'objects'. The skeleton cell data structure is responsible for collecting this information. The floorplaning process is an interactive process. In each case, the system checks user input data on compatability with its own stored-in knowledge. If the user requests an action resulting in an illegal state, the system will reject the action. In any case additional knowledge may be added at each time to improve the system. The floorplaning process is an initial step, to guide the following steps during layout phase.

- **Placement and routing of components**
  In contrast to digital placement considerations factors as heat feedback, matching, wire impedance, etc. play an important role during the placing and routing process of components. It is very difficult to decide which effects are more important or not in a certain design. The relative importance given above varies from one design to the next.
  In our design system we distinguish between wires and components, connected by wires. Wires, as opposed to circuit components, are generated by the system on demand by the user. On user prompt, connected components on the schematic view are connected in the layout view. The connection process is automatic, but the user may influence it. Control of the routing process may be

direct, indirect or both. Direct control means the user makes suggestions to or places constraints on the routing path. Indirect control means the user places constraints on the types of signals and/or components which the wire may lie parallel or cross. Constraints may be further inherited from the schematic.

The routing process entails analysis software which on-line calculates characteristics of the wire being produced. For each wire, for example, a function is called which calculates its resistance. Such parasitic values are available at any time to the user and may be included in the netlist for use in simulations.

The placement and routing process is mechanized via the Skeleton data structures. Design rule correct layout in any case is guaranteed, as are layouts which are one-to-one mappings of the schematic. So, no design rule checker or electrical rule checker is necessary.
Even if knowledge about the additional effects in analog integrated circuits like matching as mentioned above is not stored-in in the system, the circuit will work correctly in a conventional sense, a valuable feature in any design tool.

- **Moving placed components**
  Components which have been placed and then routed, will often need to be moved. The system provides a feature, where the user is relieved from disconnecting connected wires and recalculating parasitics. This is done automatically.

- *Initializations*:
  1. *schematic*
  2. *transistor width, length*
  3. *gate, source and drain pins, junctions and*
  4. *DRC informations like overlaps.*
- *Generation of graphic object primitives like rectangles etc.*
  1. *gate area (polysilicon)*
  2. *drain area (n diffusion)*
  3. *source area (n diffusion)*
- *Generate active area (conjunction of poly and 'diffusion' area)*
- *Generate the layout ports*

Fig.5 Structure of a NMOS Transistor Generator

2.2.3 Library

As pointed out in the sections above, a lot of information about the design process must be stored within the system. Not only technology information like design rules must be available for the system, but a lot of knowledge how to fullfill the demands on the circuit must be present for the system. So, we have to distinguish between at least two types of technology information within the library: the first one, direct dependent on the process to avoid fabrication errors and the second one special information on how to get the wanted result regarding mismatching, heat effects or for example other parasitic effects like crosstalking, noise etc. The information set is directly changeable by the user. The first set may be exchanged more often whereas the rule set of the second type possibly only will be supplemented by new rules.
The cell generators are integrated part of the library. They are stored in a very effective way in order to guarantee fast access time and a high degree of flexibility to the cells. This is achieved by using COMMON LISP structures.
To have access to simulation results of prior simulation runs during a design process, waveforms of for example node voltages are stored within the library ,too. Of course, to get simulation output, input information, i.e. control information for the simulation run must be available. It is input through the UI and important data as for examples node presets are stored in the library. Cells are stored in the library in a coded form (cell generators). So, to input already existing cells into the library, a program will be available to convert the typical circuit description of a cell (netlist, possibly extended) into the cell generator format. To avoid storing a cell twice, a structure detecting system will be built in.
Another very important role of the library is the connection of this system for the design of analog cells into the CVS system. This is accomplished by another interface to prepare this information for the official CVS data base for integrated circuits.

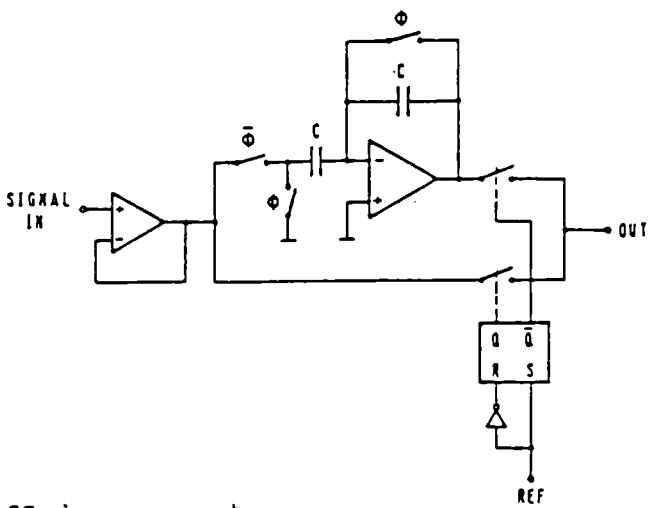


Fig. 6: SC phase comparator

2.3 Simulators and Optimizer

Integrated circuit design without verifying the results is not reasonable. So, electrical simulators are integrated in the system. The best known one is SPICE. It is usable to check many properties of a circuit like the transient or frequency behavior or to regard noise considerations of a circuit. But there are also some difficulties. If the circuit becomes larger, i.e. more than some tens of transistors, the simulation will take a lot of CPU time. By this reason, some effort is spent to develop a feature to save computing time. This is reached by the introduction of macromodels. Macromodels are well situated to model the behavior of a much more complex circuit by replacing components or groups of components by for example simple controlled sources and impedances. By doing that, simulation time goes down significantly without loosing accuracy. It must, however be mentioned, that no new macromodels are developed. Instead, already existing ones are suggested to be used dependent on their special properties to model the behavior of a certain circuit.
The second integrated simulator is BONSAI. It has similar properties as SPICE. The advantage of BONSAI over SPICE is the close connection to an existing experienced and well working CMOS process line.

The simulators mentioned above, are best suited for simulation of smaller electrical circuits, consisting of only some tens of (nonlinear) components. DOSCA, a simulator for switched-capacitor circuits reduces this problem for part. By partitioning the whole complex circuit into parts activated by phase signals at certain determinated timepoints (see fig. 6), a breakdown into different independant subcircuits is possible.
By this kind of partitioning simulation of even larger circuits is feasable. Additional to these possibilities DOSCA is extended to handle nonlinear SC-networks, too. That means, simulation of e.g. AD-Converters and PLLs is possible.
The ECONOMIC optimizer is another very useful tool to be enhanced within this project. Proper values of designable parameters, such as supply voltages and stripe widths, must be determined to fulfill the requirements of the circuit characteristics. The approach is based on a boundary curve used as an assessment criterion to visualize problem diagnosis and to control the interactive optimization process.
This program will be used, if a cell in the library meets the requested demands for part and there has been left a variable part within the cell for optimization purposes.

3 Conclusion

In the previous sections a system for analog integrated circuit design based on artificial intelligence techniques has been presented. The method to come there is a new one and allows the user to generate an analog integrated circuit starting from scratch in a very efficient individual way. Already fixed unchangeable skeleton cells are unknown within this system. They are active data structures, used to enable this flexible design style.

Based on the knowledge stored within the system, the user is guided through the whole design process. This is the reason, why also not very experienced peoples may use the system to generate analog circuit of high quality and performance.
Only very few tools are needed for design support. These are very common electrical simulators like SPICE, BONSAI and DOSCA. As has been pointed out, other tools like DRC or ERC for example are not needed, checks on correctness of the produced layout are performed implicitly by application of the system suggested design steps. The complete design package is part of the CVS project. It will be integrated into the whole CVS system through an interface from the library to the CVS data base. Output of the analog design system will be an extended layout description (possibly CIF) containing the outline, the ports to the world and the layout of the circuit itself. The system will be available for both BIPOLAR and CMOS design. First versions of all parts of the design package are running for BIPOLAR design, CMOS will follow. With the integration of comfortable digital and analog design aids for integrated circuits we will be able to realize hybrid circuits. The application area will be very large. One example may be to move intelligence from a central processor to the front end of a controlled system. Both, performance and reliability of the resulting system will increase on a large scale.

Acknowledgement

The author is pleased to acknowledge the valuable discussions with I. Rugen and D. Höppner, AEG; L. Moore, ANACAD; J. Büddefeld, W. Brokherde and B. Hosticka, FHG-IMS. W. Borutzky and B. Schwarz read the draft of this paper and gave valuable comments. I also thank K. Wölcken for his encouragement to prepare this paper and implementing this project.

References:

[1] T. J. Kowalski
An Artificial Intelligence Approach to
VLSI Design
Kluwer Academic Publishers , 1985

[2] G.L. Steele
Common Lisp, the Language
Digital Press, 1984

[3] Y. Tsividis, P. Antognetti
Design of MOS VLSI Circuits for Telecommunications
Prentice-Hall, Inc. , 1985

[4] K. Antreich, S. Huss
Ein interaktives Verfahren zur Optimierung integrierter
Schaltungen
Sonderdruck aus Electronics and Communication, Band 32,
1982

[5] E.E.E Hoefer, H. Nielinger
Spice, Analyseprogramm fuer elektronische Schaltungen,
Benutzerhandbuch mit Beispielen
Springer Verlag , 1985

[6] Alan B.Grebene
Bipolar and MOS Analog Integrated Circuit Design
John Wiley & Sons, 1984

[7] P.R. Gray;D.A. Hodges; R.W. Broderson
Analog MOS Integrated Circuits
IEEE Press, 1980

[8] Multiple Storage Quad Trees: A Simpler Faster
Alternative to Bisector List Quad Trees
IEEE Transactions on Computer-Aided-Design
Vol. CAD-5, No. 3, July 1986

[9] Second Half Year Report, EEC CVS Project, No. 802,
Work package 3, 1987

Project No. 962

Three-Dimensional Algorithms for a Robust and Efficient Semiconductor Simulator with Parameter Extraction

Giorgio Baccarani

Dipartimento di Elettronica, Università di Bologna
viale Risorgimento, 2 – 40136 Bologna, Italy

Abstract

In this paper we describe the aims, the structure and the achievements of the ESPRIT project no. 962E-17 entitled: "Three-Dimensional Algorithms for a Robust and Efficient Semiconductor Simulator with Parameter Extraction". The project started in April 1986, and is therefore at an early stage of development. Yet, very promising results have already been achieved in several areas of numerical device simulation, including discretization schemes, mesh generation, and solution procedures. The excellent European expertise in this field promises to rapidly fill the gap with USA and Japan.

1. Introduction

Numerical simulation of semiconductor devices in two dimensions is nowadays a well-established technique for the design of advanced electronic components and processes. As device miniaturization progresses toward submicron feature sizes, however, three-dimensional effects are getting more and more important even for nominally-standard planar devices, thus making two-dimensional simulation codes inadequate for device-performance prediction. In addition, increasingly complex device geometries are being devised, such as the buried-electrode dynamic RAM cell, presently being used in high-capacity memory devices, the floating-gate EPROM cell, and the I^2L NOR gate, which are inherently three dimensional. All the above devices can only be simulated by means of three-dimensional device-analysis programs.

This project aims at the investigation of suitable algorithms for the analysis of semiconductor devices in three dimensions, and at the development of a project code implementing the most efficient of those algorithms. Most of the activity reported so far in this field has been performed in Japan: Hitachi [1], NTT [2], Oki [3], and Toshiba [4] have all developed their own three-dimensional simulators. Also, a 3-D version of the IBM's code FIELDAY has been reported [5]. It is therefore appropriate that a joint European effort be focused on the solution of the challenging technical problems associated with the development of a 3-D simulation code with unprecedented flexibility and performance.

The project code to be developed in this context is expected to handle complex device geometries, thus requiring the use of non-standard elements such as isoparametric "bricks", tetrahedra or triangular based prisms; it will allow for sophisticated physical models accounting for the most important physical mechanisms affecting device behaviour, and will be able to perform steady-state, small-signal and transient analyses.

From the numerical standpoint, one of the most important differences between two- and three-dimensional device-analysis programs stems from the massively larger number of mesh points necessary for an adequate description of a 3-D structure. Due to the inherent non-linearity of the semiconductor equations, this will require repeated solution of extremely-

large systems of linear equations (some 20,000 or more) which dictates the use of entirely new, parallel algorithms to be run on suitable machines. Also, automatic generation of 3-D meshes conforming to complex device geometries is going to be an extremely challenging task, requiring a great deal of effort.

In the next section, the structure of the project is briefly summarized. Sections 3-7 discuss the current status of the activities for each workpackage of the project. In section 8 some preliminary steady-state simulations of a non-planar 3-D MOSFET will be shown, illustrating new results on a classical 3-D problem: the narrow-channel effect. Finally, conclusions are drawn in section 9.

2. Structure of the project

The partners participating in this project are listed below:

- Rutherford Appleton Laboratory (UK)
- General Electric Co. (UK)
- Philips (The Netherlands)
- SGS Microelettronica (Italy)
- Analog Devices (Ireland)
- IMEC (Belgium)
- Trinity College Dublin (Ireland)
- University College Swansea (UK)
- NMRC (Ireland)
- University of Bologna (Italy)

As can be seen, the list comprises four industrial partners, three large research laboratories and three universities. RAL is acting as a prime contractor of the project. The allocation of manpawer over the 4-year project among the various partners is shown in table I.

RAL (UK)	16 ManYears
GEC (UK)	14 ManYears
Philips (The Netherlands)	16 ManYears
SGS (Italy)	8 ManYears
ADBV (Ireland)	4 ManYears
IMEC (Belgium)	4 ManYears
TCD (Ireland)	12 ManYears
UCS (UK)	12 ManYears
NMRC (Ireland)	4 ManYears
UBO (Italy)	8 ManYears

Table I - Allocation of manpower

Five main areas of activity have been identified and, correspondingly, the project has been subdivided in five work packages, namely

- Physical models and validation
- Discrete problem formulation
- Mesh generation and refinement
- Solution procedures
- Project code

In what follows, we provide a general description of the aims and preliminary achievements for each of the above workpackages.

3. Physical models and validation

The first workpackage, "physical models and validation" is intended to investigate physical models for numerical simulators, analytical MOSFET models for circuit simulation, and parameter extraction techniques. Test-structure fabrication and measurements and software validation represent additional major topics of this activity. To date, the activity carried out in this context includes measurements of the multiplication factors at low electric field in bipolar transistors and comparisons among presently-available 2-D device simulators (IMEC), development of MOSFET capacitance models and capacitance measurements techniques (NMRC), investigations of the link betweeen process, device and circuit simulators (ADBV) and research on worst-case prediction of MOSFET parameters from sensitivity-analysis techniques (SGS).

4. Discretization techniques

Much work is presently being done in the field of discretization techniques, and different schemes are being pursued in order to assess, at a later stage of the project, the most successful ones. Preliminarly, a 3-D Poisson solver has been developed by different partners, and the following results have been achieved:

- A 3-D solver using isoparametric "brick" elements has been constructed and tested at Philips. It uses trilinear finite element discretization, ICCG to solve the linear equations, a detailed line search technique to solve the non-linear equations and can cope with large problems (20–40,000 unknowns) by suitable memory-management techniques. Degenerate bricks, i.e. bricks where corner nodes coincide have been accommodated using a Gauss quadrature rule, thus allowing flexible geometries to be described.
- A 3-D Poisson solver using tetrahedral elements has been constructed at GEC. Simple tetrahedral meshes have been obtained using division of finite difference meshes, and the combination of linear solver (ICCG) and non-linear line search techniques have been successfully checked on meshes of up to 10,000 nodes. The calculation of nodal volumes for charge lumping and pipe width for the Scharfetter-Gummel approximation has been sucessfully tackled for general tetrahedra. The present limitation is related to the lack of a suitable mesh generator.
- A finite-difference Poisson solver has been constructed at TCD, which has been tested on problems up to 10,000 nodes. ICCG is used to solve the equations, and machine virtual memory is used to accommodate the problem size. More recently, this has been extended to the on-state problem.
- A finite difference Poisson solver has been constructed at UCS and tested on problems up to 1,000 nodes. Lumped charge, a direct solver, and a non linear line search technique are currently being used. The limitation is currently placed by availability of mesh generation, and the expansion to larger problems will require a change to an indirect solver (ICCG).
- A 3-D solver using triangular prisms has been succesfully constructed at the University of Bologna. It combines Poisson solution and one carrier (electron) continuity equation, which makes it suitable for a nearly exhaustive simulation of unipolar devices. The problem size is currently limited to a few thousand nodes as a direct solver is used for the linear equations. The line search method by Bank and Rose [6] is successfully employed by the Poisson solver, and a convergence acceleration algorithm first suggested by Anderson [7] has been implemented to speed up the convergence rate of

the employed decoupled scheme. Current developments include a change of the linear solution technique.

Quite a bit of effort has been devoted by Philips to the extension of the Sharfetter-Gummel [8] discretization scheme to quadrilateral elements in two dimensions. Boxes are constructed using the centroides of the elements and the midpoint of the element faces and edges. An expression for the distribution of the current density in the element has been obtained, and the tangential component along mesh edges matches the one-dimensional Scharfetter-Gummel formula. This approach is going to be extended to the third dimension in the near future.

A final remark is in order at this point to better clarify how the development of different Poisson solvers will contribute to the design and implementation of the final code. Due to the lack of previous experience in 3-D simulation, several options had to be explored before taking final choices: type of elements, discretization schemes, linear solvers, etc. It was felt that this kind of activity could be suitably subdivided among the partners according to their own specific interests, and that creativity had to be encouraged rather than suppressed at this stage of the project. This preliminary work is therefore of the utmost importance to assess the relative advantages and disadvantages of the various numerical techniques, and will help selecting the most appropriate ones for the final 3-D code.

5. Mesh generation and refinement

Four major areas of activity have been identified:

- Production of tetrahedral meshes by the generalization of the 2-D Delaunay method;
- Generation of prismatic meshes by extending the capability of existing two-dimensional triangular generators into three dimensions;
- Mesh generation and refinement using hexahedral elements;
- Use of multigrid techniques in device simulation.

There are several methods of generalizing Delauney mesh generation to 3-D problems. The following techniques have been considered

a) Generation using an underlying hexahedral mesh;
b) Generation using only a boundary description of the domain;
c) Generation using the convex hull of a transformed coordinate space.

Approach (a) is of course the simplest, but it does not allow for the exploitation of the potential generality of tetrahedral elements. Experience to date with tetrahedral mesh generation indicates that there are a number of additional logical problems present in three dimensional generation which are not found in two dimensions. These are in the areas of element degeneracy and topology swapping to form Delauney tetrahedra.

Two partners (UCS and UB) have generalized existing two-dimensional mesh generators by projecting two-dimensional slices through the device, and by merging the projections. This technique leads naturally to the use of prismatic elements in the discretization. These extensions are based on the two-dimensional generators GENTIP (UCS) and ATMOS (UB). Considerable success has already been achieved with this type of mesh generation. Figure 1 shows a typical 3-D MOSFET mesh generated at the University of Bologna. As can be seen from the figure, the use of prismatic elements does not prevent the user from handling non-planar structures, although geometrical non-uniformities in the third dimension can only be taken care of by a step-like boundary.

Work in the area of hexahedral mesh generation is currently being pursued at Philips, where

the use of quadrilateral meshes in 2-D has long been advocated.

6. Solution procedures

This workpackage aims at providing robust and efficient algorithms for the solution of the 3-D semiconductor equations. It is subdivided in the following tasks

a) Linear solvers;
b) Non-linear solvers;
c) Transient solvers.

Concerning point (a) above, two basic methods can be used for the solution of the linear system, namely, i) direct methods and, ii) preconditioned iterative methods. The choice af an appropriate method is essential to ensure numerical efficiency of the code, as most of the required CPU time is spent for the repeated solution of the linear system. Depending upon the properties of the coefficient matrix, different solution methods must be selected. For the Poisson problem, the resulting coefficient matrix is symmetric and positive definite. In this case, it is generally accepted that the conjugate gradients method, combined with an incomplete Choleski factorization (ICCG), is more efficient than direct methods, at least for a large number of equations.

For the linear system occurring during the solution of the full set of equations, no definite conclusion can be drawn. In this case the involved matrices are not symmetric, and the ICCG method is ruled out. Therefore, either L-U decomposition or a variant of the CG method, the so-called "Conjugate gradients squared" (CGS) method, has to be used in connection with an incomplete block preconditioning [9]. In the latter case, however, the preconditioning depends heavily on the regular structure of the matrix, which cannot be ensured with tetrahedral and triangular-based prismatic elements, or when adaptive refinement of the mesh is performed during computation.

For the solution of the linear system, a speed-performance advantage can be achieved using vector processors and/or parallel architectures. Consequently, suitably parallel algorithms will have to be developed, even though the portability of the code will be severely restricted. It is felt that some of the most innovative results of this project are to be expected in this area, which is far less explored than others highlighted in the previous sections.

7. Specifications of the project code

The design and implementation of a 3-D project code is perhaps the most challenging task of this project, as it involves a cooperative effort from several partners and a clear definition of suitable interfaces between input preprocessor, mesh generator, numerical solver, and output-graphics facilities. In order to comply with the above requirements, the use of a suitable data-base management system is in order.

The input preprocessor is intended to provide a complete description of the geometrical and physical structure to be simulated, a definition of the physical parameters pertaining to the various device regions, and the required commands for the solver, including bias points for DC and AC solutions and applied waveforms for transient analyses.

The mesh generator ought to be, as far as possible, automatic, in order to relieve the user from the need of an interactive refinement, which requires a great deal of physical understanding of the device behaviour.

The main properties of the numerical solver are listed below

- Geometrical flexibility, to make it suitable for any kind of devices (MOSFET's, bipolar and power devices);
- Availability of relevant physical models, such as SRH and Auger recombination, band-gap narrowing, impact ionization, etc.;
- Availability of a number of boundary conditions to allow for ideal and resistive ohmic contacts, Schottky-barrier junctions, gates and floating gates;
- Availability of DC, AC and transient analyses;
- Availability of voltage, current and mixed boundary conditions on both ohmic and Schottky contacts;
- Adaptive-mesh refinement capability.

Finally, sophisticated graphic tools intended to visualize several output data, including unknown functions in a 3-D domain, perspective contour surfaces, 3-D plots on arbitrary device cross sections etc. must be provided. Here again the need of standardization requires that some graphic system, such as GKS, be selected as a common graphics standard throughout the project.

8. MOSFET simulation in three dimensions

As a first example, a classical 3-D problem, namely the narrow-channel effect in MOSFET's has been tackled at the University of Bologna using a newly-developed 3-D code called HFIELDS-3D [10]. The discretization mesh is shown in figure 1. The simulated MOSFET has a channel length $L = 1\,\mu m$ and a nominal channel width $W = 1.2\,\mu m$. Due to the symmetry of the structure, only half device is actually considered. In the front plane, where we accommodate the triangular bases of the prisms, we take advantage of the flexibility of the mesh which can conform to the "bird's beak" at the transition between the channel and the field region. Current flow occurs mainly in the third dimension, i.e. normal to the front plane.

Figure 2 shows a perspective plot of the electric potential in equilibrium in the plane at the thick-oxide silicon interface. The two upper "plateau" represent the source and drain regions, separated by a well in the channel region. Figure 3 shows instead a plot of the electric potential in the plane at the gate-oxide silicon interface. As this plane enters the thick-oxide region, we notice a ridge at the periphery of the channel and an increased potential in the field-oxide, due to the different relationship between the surface potential and the gate voltage.

The fringing effect due to the field-oxide penetrates deep in the channel region when the device is biased in subthreshold. Consequently, we notice a smaller current in the 3-D structure than we have in a corresponding 2-D MOSFET with the same nominal channel width. On the other hand, if we define an effective channel width by fitting the 3-D current in subthreshold, we end up with a smaller 2-D current in strong inversion. This effect is demonstrated in figure 4, where we compare the 2-D and the 3-D turn-on characteristics at $V_{DS} = 0.1$ V, having adjusted the effective channel width of the 2-D device. Figure 5 shows the corresponding turn-on characteristics with $V_{DS} = 3.0$ V. The increased drain voltage tends to enhance the penetration of the field lines in the channel region, thereby reducing the drain current in subthreshold even further.

From the above considerations it turns out that the fringing field which is responsible for the so called "narrow-channel" effect, is a rather complex function of the gate and drain voltage, as well as of the substrate voltage, and cannot be simply described in terms of a threshold shift. The use of a 3-D simulation code can therefore be of some help in gaining a better understanding of this effect, and it can represent a valuable alternative for predicting

the electrical characteristics of narrow-channel MOSFET's.

9. Conclusions

In this project the problem of investigating the most efficient algorithms for 3-D simulation of semiconductor devices has been tackled. A number of discretization schemes is being pursued by various partners, and the obtained results will soon be compared in order to identify the most successful ones to be incorporated in the project code. Results achieved after a one-year work are very encouraging, as a preliminary version of a general-purpose 3-D code using triangular-based prismatic elements has already been demonstrated, and a Poisson solver based on tetrahedral elements has been successfully constructed. Triangular prisms and tetrahedra are at the present time the most promising elements to be considered for future developments. From the standpoint of geometrical flexibility, the latter should be preferred. However, the problem of appropriately defining the control volumes to be associated with each node in the general case of a tetrahedral mesh is at the present time largely unexplored and, in the author's view, the solution of this problem will be the key to the success of the method.

Much work remains to be done in several areas in order to produce a complete simulation system to be used as a design tool in an engineering environment: among these, a user-oriented input preprocessor, a reliable and flexible tetrahedral-element mesh generator and a powerful graphics system for the visualization of the output data. Also, from the algorithmic standpoint, work must be done in order to improve the efficiency of the linear solver to be used when the coefficient matrix is non-reciprocal, as it happens to be when a coupled solution scheme of the basic equations is used. Recent results obtained at Philips with the CGS method combined with an incomplete L-U preconditioning appear to be very promising, while not being tied to a regular matrix pattern.

In any case, 3-D simulation is going to be far more expensive than 2-D simulation, owing to the extremely large number of mesh points required for a detailed description of a 3-D semiconductor device. Consequently, parallel algorithms will have to be developed in order to reduce the computation time below acceptable limits. It is felt that the most innovative results, as far as efficiency is concerned, will probably come from this area of activity, as a major breakthrough is unlikely to emerge from new studies in the field of numerical techniques.

Acknowledgements

The author is indebted to A. Bryden (RAL) and P. Mole (GEC), who provided informative material on the current status of the activities within the project. Figures 1–5 are the result of a cooperative work carried out at the University of Bologna. Finally, financial support from the European Community is gratefully acknowledged.

References

1. T. Toyabe, H. Masuda, Y. Aoki, H. Shukuri and T. Hagiwara: "Three-dimensional device simulator CADDETH with highly convergent matrix solution algorithms", *IEEE Trans. on Electron Devices*, vol. ED-32, pp. 2038-2043, 1985.
2. K. Yokoyama, M. Tomizawa, A. Yoshii and T. Sudo: "Semiconductor Device Simulation at NTT", *IEEE Trans. on Electron Devices*, vol. ED-32, pp. 2008-2017, 1985.
3. Y. Namba, J. Ueda, T. Miyoshi and S. Ushio: "Two/Three Dimensional Device Simulator", from *Simulation of Semiconductor Devices and Processes*, Eds.: K. Board and D. R. J. Owen, Pineridge Press, Swansea, 1986.

4. N. Shigyo and R. Dang: "Three-Dimensional Device Simulation Using a Mixed Process/Device Simulator", from *Process and Device Modeling*, Ed.: W. L. Engl, North Holland, 1986.
5. E. M. Buturla, P. E. Cottrell, B. M. Grossman, C. T. McMullen and K. A. Salsburg: "Three-Dimensional Transient Finite-Element Analysis of the Semiconductor Transport Equations", from *Numerical Analysis of Semiconductor Devices, Proc. of the NASECODE II Conference*, pp. 160-165, Boole Press, Dublin, 1981.
6. R. E. Bank and D. J. Rose: "Parameter Selection for Newton-Like Methods Applicable to Nonlinear Partial Differential Equations", *SIAM Journal of Numerical Analysis*, vol. 17, pp. 806-822, 1980.
7. D. G. Anderson: "Iterative Procedure for Nonlinear Integral Equations", *A.C.M. Journal*, vol. 12, pp. 547-560, 1965.
8. D. L. Sharfetter and H. K. Gummel: "Large-Signal Analysis of a Silicon Read Diode Oscillator", *IEEE trans. on Electron Devices*, vol. ED-16, pp. 64-77, 1969.
9. J. Meijerink: "Iterative Methods for the Solution of the Linear Equations Based on Incomplete Factorization of the Matrix", *Publ. 643, Shell*, Rijswijk, The Netherlands, 1983.
10. P. Ciampolini, A. Gnudi, R. Guerrieri, M. Rudan and G. Baccarani: "Three-Dimensional Simulation of a Narrow-Width MOSFET", to be presented at the *ESSDERC-87*, Bologna, Sept. 14-17, 1987.

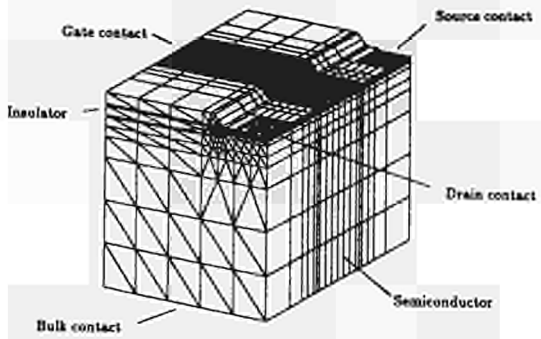


Fig. 1: Prismatic-element 3-D mesh of the simulated MOSFET.

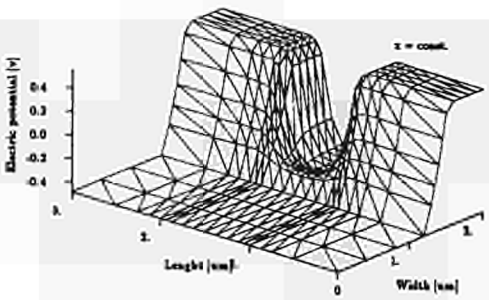


Fig. 2: Perspective plot of the electric potential in the plane at the field-oxide silicon interface.

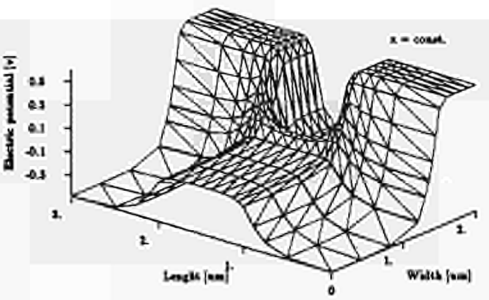


Fig. 3: Perspective plot of the electric potential in the plane at the gate-oxide silicon interface.

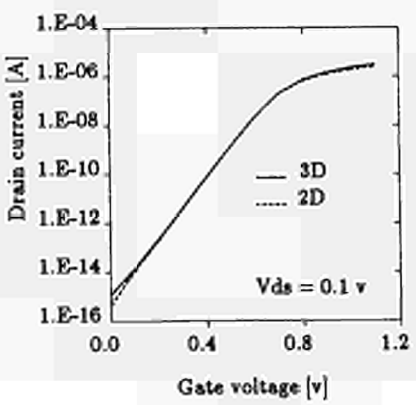


Fig. 4: MOSFET turn-on characteristics according to the 2-D and 3-D simulation codes. The drain voltage is V_{DS} = 0.1 V.

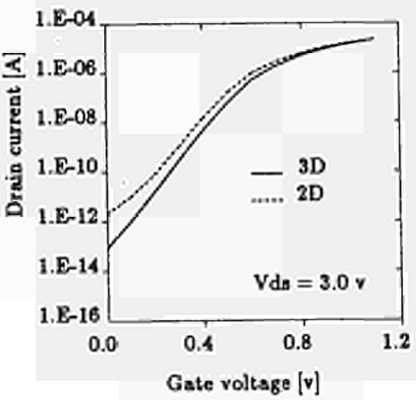


Fig. 5: MOSFET turn-on characteristics according to the 2-D and 3-D simulation codes. The drain voltage is V_{DS} = 3.0 V.

II. SOFTWARE TECHNOLOGY

Parallel Sessions

Project No. 937

SPECIFYING MESSAGE PASSING AND REAL-TIME SYSTEMS WITH REAL-TIME TEMPORAL LOGIC

Ron Koymans & Ruurd Kuiper

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

Erik Zijlstra

Foxboro
Koningsweg 30, 3762 EC Soest, The Netherlands

Abstract

Temporal logic is a simple extension of classical logic with temporal operators. When a computation is seen as a sequence of states changing over time, one can reason about such sequences with temporal logic: the classical part describes the static aspect, the states, and the temporal operators describe the dynamic aspect, the relation (in time) between states.

Temporal logic has proved to be a most versatile tool for the specification and verification of concurrent systems. It has been applied to a wide variety of systems, such as concurrent programs, communication protocols, hardware, VLSI etcetera.

Nevertheless, temporal logic is not suited on beforehand for the specification of two important classes of systems: message passing and real-time systems. We introduce an assumption that enables the description of message passing systems and develop an extension of standard temporal logic, called real-time temporal logic, for describing real-time systems. We illustrate the resulting formalism by three examples.

This work was carried out as part of ESPRIT project 937:
Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES).

1. INTRODUCTION

Temporal logic is a simple and elegant extension of classical logic (propositional and predicate logic) with temporal operators for reasoning about situations changing in time. The underlying semantics of temporal logic makes a clear distinction between the static aspect of a situation, represented by a state, and the dynamic aspect, the relation (in time) between states. This distinction is reflected in the syntax: a state is described by classical logic, while the temporal operators are used for the description of the evolution of the situation over time. In this way states and time need not be introduced explicitly in the logic itself.

This picture of states and their relation in time fits well with the notion of computation as used in computer science. A computation can be seen as a sequence of states where each transition from one state to the next state in the sequence (each step of the computation) can be thought of as a tick of some computation clock. The corresponding time model is in that case the set of natural numbers. Temporal logic with the natural numbers as time domain is a variant of linear time temporal logic. This variant is especially suited for the description of computer systems which are then seen as generators of execution sequences.

Since the introduction of (linear time) temporal logic in the area of program verification ([P1]), it has proved to be a most versatile tool for the specification and verification of concurrent systems. In that context it can, amongst others, be used for

- reasoning about safety properties (e.g. partial correctness, mutual exclusion) and liveness properties (e.g. total correctness, fairness),
- describing systems at any level of abstraction,
- compositional reasoning: the specification of the whole system is a function of the specifications of its subcomponents.

Consequently, temporal logic has numerous applications in computer science. It has been successfully applied as a specification and verification method for concurrent programs, communication protocols, VLSI, hardware etcetera. It can furthermore be used to give axiomatic definitions of concurrent programming languages and some temporal logics even have been made executable (thereby unifying programs and specifications). More information, including further references, can be found in overview papers by Lamport ([L1]) and Pnueli ([P2]).

Nevertheless, temporal logic is not suited on beforehand for the specification of two important classes of systems: message passing and real-time systems. The importance of these two classes is stressed by their manifold appearances in practice:

- message passing is one of the most important means of interprocess communication in distributed systems, either on a high level (e.g. in telecommunication applications where programming could be done in a high-level concurrent language with asynchronous message passing such as

CHILL [CHILL]) or on a lower level (such as in implementations of synchronous languages for distributed computing like Ada [Ada]),

- among the many real-time applications (e.g. on-line reservation systems) there are some highly critical systems such as computer controlled chemical plants and nuclear power stations.

Because message passing systems are so widely used and the dangers of malfunctioning real-time systems affect most of us (think e.g. of flight control software for civil airplanes), it is of *vital* importance to develop formal techniques for reasoning about them. For message passing this development has been actively going on for several years. For real-time, however, the situation is alarming: theoretical research has almost completely ignored real-time aspects.

In this paper we introduce an assumption that enables the description of message passing systems and give an extension of standard temporal logic, called real-time temporal logic, for the description of real-time systems. We illustrate the resulting formalism by three examples: a classification of pure message passing systems, a pure real-time system and a mixture of both.

This paper is structured as follows. In section 2 we give a short summary of propositional temporal logic. Section 3 contains a description of message passing and real-time systems. Real-time temporal logic is introduced in section 4. Section 5 contains the three examples mentioned above.

2. PROPOSITIONAL TEMPORAL LOGIC

We first define the syntax of PTL, Propositional (Linear Time) Temporal Logic.

Let I be a non-empty set.
PTL uses a propositional language with

Vocabulary: atomic propositions $P_i (i \in I)$
propositional connectives $\neg$, $\wedge$
temporal operators X, U, Y, S

Formulas: $P_i (i \in I)$
$\neg f_1, f_1 \wedge f_2, Xf_1, f_1 U f_2, Yf_1, f_1 S f_2$ (f_1, f_2 formulas).

The operators X, U, Y, S are called respectively 'next-time', 'until', 'last-time' and 'since'.

The semantics of PTL is as follows. A state is a mapping from I to $\{True, False\}$: a state indicates which atomic propositions are true in that state. A model is an infinite sequence of states. An interpretation is a pair $<M, n>$ where M is a model and n a natural number (representing the present). Truth of a formula f in an interpretation $<M, n>$, notation $M, n \models f$, is inductively defined by:

$$
\begin{aligned}
M, n \models P_i \quad &:= \quad M_n(i) = \mathit{True} \\
M, n \models \neg f \quad &:= \quad \mathit{not}\ M, n \models f \\
M, n \models f_1 \wedge f_2 \quad &:= \quad M, n \models f_1 \ \mathit{and}\ M, n \models f_2 \\
M, n \models Xf \quad &:= \quad M, n+1 \models f \\
M, n \models f_1 U f_2 \quad &:= \quad \text{there exists } m \geqslant n \text{ such that } M, m \models f_2 \text{ and} \\
&\qquad \text{for all } j \text{ such that } n \leqslant j < m: \quad M, j \models f_1 \\
M, n \models Yf \quad &:= \quad n > 0 \ \mathit{and}\ M, n-1 \models f \\
M, n \models f_1 S f_2 \quad &:= \quad \text{there exists } m \leqslant n \text{ such that } M, m \models f_2 \text{ and} \\
&\qquad \text{for all } j \text{ such that } m < j \leqslant n: \quad M, j \models f_1.
\end{aligned}
$$

From the four temporal operators above many other temporal operators can be derived. Two very important temporal operators are F ('eventually') and its dual G ('henceforth'). These can be defined by

$Ff := \mathbf{true}\, Uf$ where $\mathbf{true} \equiv \neg (P_i \wedge \neg P_i)$ for some $i \in I$,

$Gf := \neg F \neg f$.

As an example of the use of these operators we mention the combination $GF...$ which corresponds intuitively with 'infinitely often'. This combination is often used for the expression of fairness properties.

3. MESSAGE PASSING AND REAL-TIME SYSTEMS

Let *Messages* be a non-empty set of messages. A schematic picture of a message passing system could be

MPS = Message Passing System

where $m \in \mathit{Messages}$ and

in (m) corresponds to the acceptance (from the environment) of message m by the MPS, and

out (m) corresponds to the delivery (to the environment) of message m by the MPS.

The MPS can be a simple buffer or transmission medium but also a complex communication network. *in* (m) and *out* (m) constitute the interface with the environment and *out* (m) is considered to be the system reaction on the environment action *in* (m). Of course, the above picture should be supplemented by restrictions on the functions *in* and *out*, dependent on the particular type of message passing system considered. For all types we take the following restrictions as basic assumptions:

BA1. the acceptance and delivery of messages can be viewed as instantaneous actions (in the sense that always a unique moment of time can be identified at which a message can be said to be accepted, respectively delivered), which are always possible,

BA2. at any moment of time, at most *one* message can be accepted (respectively delivered),

BA3. the MPS does not create messages by itself (in other words: the bag of delivered messages is always *some part* of the bag of accepted messages),

BA4. the speed of the MPS is finite, i.e. there is a positive (maybe infinite) delay between the acceptance of a message and its delivery.

An example of a MPS often occurring in practice and satisfying BA1-BA4 is a transmission medium with a probability between zero and one of a successful transmission.

Besides the basic assumptions above, message passing systems can be distinguished further by properties such as

- reliability properties
  - perfect: all accepted messages are (eventually) delivered
  - imperfect: messages may get lost
- ordering disciplines
  - FIFO (like a queue)
  - LIFO (like a stack)
  - unordered (like a bag).

An example of an unordered MPS is a communication network in which every message is sent on to an arbitrary node until it reaches the destination node.

It is difficult to list precise characteristics for real-time systems like we gave above for message passing systems. However, time clearly does play a dominant role. Three subjects we want to mention here are:

- 'promptness requirements', e.g. every time A occurs, B must follow within 3 seconds
- 'periodicity properties', e.g. A occurs regularly with a period of 3 seconds
- time models: some real-time systems control continuously changing physical entities, such as volume and temperature; in such a case a discrete time model (like the natural numbers) is questionable.

4. REAL-TIME TEMPORAL LOGIC (RTL)

It can be proved (see [K]) that many message passing systems can not be specified with PTL (see section 2) or even its first-order extension. The essential

fact here is that PTL can not distinguish the different instances of one and the same message accepted by the MPS and hence can not trace back (in time) every delivered message to its *unique* moment of acceptance. For real-time systems there is a simpler reason why these can not be specified with standard temporal logic: PTL has only qualitative temporal operators and hence is not capable to express quantitative measures of time. Furthermore, the semantics of PTL is based on a discrete time model and, as noted at the end of the previous section, this complicates the description of specific applications in which continuous processes are involved.

To solve the problems for message passing systems we assume that incoming messages can be uniquely identified, e.g. by means of *conceptual* time stamps. This makes the above mentioned tracing of delivered messages to their moments of acceptance possible. The assumption of unique identification is not as restrictive as it may seem on first sight. This assumption can be justified by the notion of data-independence of [W]. Informally, a system is called data-independent when the values of the supplied data do not influence the functional behaviour of the system. One of the results of [W] implies that the correctness of a data-independent system does not depend on the uniqueness of the incoming data. Since message passing systems only *pass* data, they are clearly data-independent. The problems for real-time systems lead to our motivation for RTL, Real-Time Temporal Logic. RTL introduces quantitative temporal operators and its semantics allows dense time models also, like the rational and real numbers.

We now define RTL more formally. Consider a time domain T with a linear order $<$ (in particular we think of the natural and real numbers). RTL uses a first-order language with quantification over:

- the data domain, e.g. *Messages* : $\forall m, \exists m$,
- the time domain T : $\forall t, \exists t$.

Note that quantification ranges over global variables only.
The temporal operators are $U_{=t}$ and $S_{=t}$ for all $t \in T$.

The semantics of RTL is as follows. Let Σ be the set of all states. A model is a mapping from T to Σ. For an interpretation $<M, t>$, where M is a model and $t \in T$, the temporal operators are defined as follows:

$$M, t \models f_1 U_{=t_0} f_2 := M, t+t_0 \models f_2 \text{ and for all } t' \text{ such that } t < t' < t+t_0 : M, t' \models f_1$$

$$M, t \models f_1 S_{=t_0} f_2 := M, t-t_0 \models f_2 \text{ and for all } t' \text{ such that } t-t_0 < t' < t : M, t' \models f_1.$$

In the above definitions it is assumed that $t+t_0$ and $t-t_0$ exist in T. Whenever this is not the case (e.g. $t-t_0<0$ for the natural numbers as time domain) the above formulas are false in $<M, t>$.

Using these real-time operators again many derived operators can be defined amongst which the original U and S of PTL from section 2:

$$f_1 U f_2 := f_2 \vee (f_1 \wedge \exists t\ (t>0 \wedge f_1 U_{=t} f_2))$$

$f_1 S f_2 := f_2 \vee (f_1 \wedge \exists t\ (t>0 \wedge f_1 S_{=t} f_2))$.

Apart from the operators F and G as defined at the end of section 2 we also use an analogue of F that refers to the past instead of the future, but without including the present:

$Pf := \exists t\ (t>0 \wedge \mathbf{true}\, S_{=t} f)$.

5. EXAMPLES

5.1 Message Passing Systems

Our first example concerns message passing systems. Under the assumption of uniqueness of accepted messages, which can be translated as

$G \forall m \neg (in(m) \wedge P\, in(m))$,

we can formulate the basic assumptions BA1-BA4 from section 3 as the following set of axioms:

BA 2 $\quad G \forall m \forall m' [((in(m) \wedge in(m')) \vee (out(m) \wedge out(m'))) \rightarrow m=m']$

BA 3a,4 $\quad G \forall m [out(m) \rightarrow P\, in(m)]$

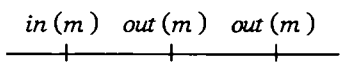
BA 3b $\quad G \forall m \neg (out(m) \wedge P\, out(m))$.

There is no need to specify BA1 because this is already fulfilled by the nature of the formalization: $in(m)$ and $out(m)$ can be true or false at any moment. Notice that we split BA3 (no creation of messages) into the following two cases:

BA3a no creation of altogether *new* messages,

BA3b no *multiplication* of messages already present.

Axiom BA 3a,4 does not cover requirement BA3b as is shown by the BA3b-illegal behaviour



which is allowed by this axiom. Therefore we need a separate axiom BA 3b.

Next we specify FIFO, respectively LIFO:

FIFO $\quad G \forall m \forall m' [(out(m) \wedge P\, out(m')) \rightarrow P\, (in(m) \wedge P\, in(m'))]$

LIFO $\quad G \forall m \forall m' [(out(m) \wedge P\, out(m')) \rightarrow$
$\qquad (P\, (in(m') \wedge P\, in(m)) \vee P\, (out(m') \wedge \neg P\, in(m)))]$.

Both axioms are independent of the loss of messages, in other words of the perfectness of the MPS. FIFO simply says: if m comes out after m', then m must also have come in after m'. LIFO distinguishes two cases when m comes out after m':

1. m' was put on the stack when m was already there
2. m' was already taken from the stack before m was put on it.

Note that FIFO and LIFO become equivalent when it is additionally assumed that the capacity of the message passing system to store messages is 1 (since in that case the first disjunctive clause of LIFO, point 1 above, is impossible). It is easy to check that the axiom for either FIFO or LIFO together with the assumption about the uniqueness of incoming messages imply axiom BA 3b.

Intuitively, all the formalized properties above are safety properties. It is nice to notice that all axioms above use only the temporal operators G and P and hence are safety properties according to the syntactical characterization of temporal formulas into safety and liveness properties of [LPZ]. When we want to formalize a typical liveness property such as being perfect the corresponding axiom uses the liveness operator F:

$$G\ \forall m\ [in(m) \rightarrow F\ out(m)].$$

Although the above specification is quite formal, we like to draw attention to an alteration that enables a clearer distinction between the roles of component and environment. The properties formulated in the axioms about message passing systems constitute in fact requirements about both the MPS itself and its environment. Consider, for example, BA3a,4 respectively uniqueness of messages. Also, some of the properties are, at least in their formulation, interdependent. For instance, nonduplication of messages by the MPS can only be formulated in the manner of BA3b by virtue of the uniqueness of messages property of the environment. Furthermore, the aim is in general to specify and construct a component, rather than its environment. Although in the above description of message passing systems it is fairly easy to see which responsibilities are intended to be fulfilled by the component, this is not always the case. (Even in this simple example a mischievous interpretation is allowed in which the MPS does nothing at all, but where the environment miraculously sends messages on the out line as well, in just the required manner.)

It pays therefore to give specifications in such a manner, that it is precisely the component properties that are requested, described via the dependencies upon environment properties. A formalism which allows to express the distinction between component and environment activity is then necessary. We give an outline of how this can be achieved, exemplified on the MPS treatment above. More information about this approach can be found in, e.g. , [L2,BKP].

Firstly, we view specified behaviour as given in terms of a slightly extended interface. In the MPS example above we used predicates $in(m)$ and $out(m)$. We extend this interface with the information of who is active: component or environment (or both). We express this distinction by means of extra predicates $act(C)$ and $act(E)$, denoting activity of component, respectively environment.

Secondly, to emphasize the component/environment distinction, we usually describe the requirements about the component and its dependency on the

behaviour of arbitrary environments via an implication of the form: environment properties → component properties. Note, that although in most cases this suffices, sometimes the dependencies are of a more intricate form than captured by the implication. Then this restriction can be dropped but the idea of describing the component in an arbitrary environment should be maintained: it is not allowed to forbid certain environments, but components can be allowed to misbehave in unsuitable environments.

The specification of the MPS example is then adapted as follows. We assume a model in which at each moment environment, component or both are active. This is expressed by

$$G\ (act(C) \vee act(E)).$$

The MPS is then specified by:

If {the environment satisfies}

$G\ \forall m\ \neg\ (in(m) \wedge P\ in(m))$
{uniqueness of messages}
and
$G\ \forall m\ \forall m'\ [(in(m) \wedge in(m')) \rightarrow m = m']$
{no simultaneous inputs}
and
$G\ (out(m) \rightarrow act(C))$
{no output by environment}

then {the component satisfies}

$G\ \forall m\ \forall m'\ [(out(m) \wedge out(m')) \rightarrow m = m']$
{no simultaneous outputs}
and
$G\ \forall m\ [out(m) \rightarrow P\ in(m)]$
{no creation}
and
$G\ \forall m\ \neg\ (out(m) \wedge P\ out(m))$
{no multiplication}
and
$G\ (in(m) \rightarrow act(E))$
{no input by component}.

Note, that in an environment which does misbehave, e.g., inputs the same message more than once, the component is allowed not only to output this message more than once, but to show any aberrant behaviour. This is consistent with the idea that when applied in an inappropriate environment, a component may fail.

5.2 Watchdog Timer

Our second example specifies a pure real-time system, a watchdog timer. A processor is monitored by a timer, the watchdog. The processor sets the timer with time-out period t by a signal *enable* (t) and it should reset the timer by a reset signal each time before the time-out period expires. When the processor does not succeed in resetting the timer in time, the processor will be halted by a halt signal from the watchdog. At any time, the processor and the watchdog timer can be restarted by an initiate signal from the environment (e.g. an operator pushing a button). Once the timer is set with *enable* (t) after an initiate signal, the time-out period remains t (and thus every subsequent *enable* (t') signal is ignored) until the next initiate signal.

To identify the first *enable* (t) after an initiate we define

$$\mathit{firstenable}(t) \equiv \mathit{enable}(t) \wedge (\neg \exists t'\ \mathit{enable}(t'))\ S\ \mathit{initiate}.$$

The only essential thing to be specified is the generation of the halt signal. For the moment ignoring the possibility of an interrupting initiate signal, this can be characterized by

$$G(\mathit{halt} \longleftrightarrow \exists t\ [\, t > 0 \wedge \neg \mathit{reset}\ S_{=t}\ (\mathit{firstenable}(t) \vee (\mathit{reset} \wedge \neg \mathit{halt} \wedge \neg \mathit{halt}\ S\ \mathit{firstenable}(t)))]).$$

The explanation hereof is as follows. A halt signal may be generated if and only if the timer just timed out with some period t, so during that period t no reset signal occurred and at the start of that time-out period *either* the timer was set (for the first time after an initiate) *or* the first reset signal (since the timer was set) that is not followed by another reset within a period t occurred (to get the *first* reset signal it is required that the processor has not already been halted since the timer was set).

An interrupting initiate signal would restart the whole system and to incorporate this we have to add that during the whole period of time concerned no initiate signal occurs. This leads to the final and complete specification

$$G(\mathit{halt} \longleftrightarrow \exists t\ [\, t > 0 \wedge (\neg \mathit{reset} \wedge \neg \mathit{initiate})\ S_{=t}\ ((\mathit{firstenable}(t) \wedge \neg \mathit{initiate}) \vee (\mathit{reset} \wedge \neg \mathit{halt} \wedge \neg \mathit{initiate} \wedge (\neg \mathit{halt} \wedge \neg \mathit{initiate})\ S\ (\mathit{firstenable}(t) \wedge \neg \mathit{initiate})))]).$$

5.3 Terminal Adaptor

Our third example is a mixture of message passing and real-time. It concerns a simplified terminal adaptor. On one side bytes are received from a data link operating on 512 bytes/second. On the other side bytes are transmitted to a terminal with a rate of 300 bytes/second. The adaptor has a buffering capacity of

N_1 bytes and it prevents buffer overflow through sending stop and start signals to the data link as soon as the buffer becomes more than 80% full, respectively more than 80% empty. It is assumed that after the sending of a stop signal at most N_2 bytes are sent by the data link (of course N_2 should be less than one fifth of N_1). The data link may resume sending bytes only after it has received a start signal.

Let $in(b)$ denote the reception of byte b from the data link and $out(b)$ the transmission of byte b to the terminal. Since the terminal adaptor operates as a perfect FIFO message passing system (with additional real-time restrictions), we assume uniqueness of incoming bytes

$G\ \forall b\ \neg\ (in(b)\ \wedge\ P\ in(b))$

and hence can use the axioms for message passing systems, in particular BA 2, BA 3a,4 , FIFO and perfect:

1. $G\ \forall b\ \forall b'\ [((in(b)\ \wedge\ in(b'))\ \vee\ (out(b)\ \wedge\ out(b')))\ \rightarrow\ b=b']$
2. $G\ \forall b\ [out(b)\ \rightarrow\ P\ in(b)]$
3. $G\ \forall b\ \forall b'\ [(out(b)\ \wedge\ P\ out(b'))\ \rightarrow\ P\ (in(b)\ \wedge\ P\ in(b'))]$
4. $G\ \forall b\ [in(b)\ \rightarrow\ F\ out(b)]$

As already remarked, requirement BA3b follows from the uniqueness of incoming messages together with FIFO.

For the real-time part we need some more derived real-time operators:

$F_{=t_0} f\ :=\ \mathbf{true}\ U_{=t_0} f$

$f_1 S_{>t_0} f_2\ :=\ \exists t\ (t>t_0\ \wedge\ f_1 S_{=t}\ f_2).$

Similarly one can define $F_{<t_0}$ etcetera. Furthermore we need a temporal operator like P, but including the present:

$\tilde{P}f\ :=\ f\ \vee\ Pf\ .$

Using $\tilde{P}$ we can express that byte b is at the moment contained in the buffer of the terminal adaptor:

$buffered(b) \equiv \tilde{P}\ in(b)\ \wedge\ \neg\ \tilde{P}\ out(b).$

Just to illustrate the specification method we assume that the reception from the data link is regular, with period 1/512, while the transmission to the terminal is irregular, but whenever a byte is available a transmission takes place within 1/300. The latter can be specified by

5. $G\ [(\exists b\ buffered(b)) \rightarrow F_{<1/300}\ \exists b'\ out(b')].$

The specification of the regularity at the other side is complicated by the presence of the stop and start signals. But whenever these signals do not interfere in the period of 1/512, the reception is regular. This can be specified by

6. $G\ \forall b\ [(in(b)\ \wedge\ \neg\ F_{<1/512}\ start\ \wedge\ \neg\ F_{=1/512}\ ((\neg\ start)\ S\ stop)) \rightarrow (\neg\ \exists b'\ in(b'))\ U_{=1/512}\ \exists b'\ in(b')]$.

After the data link received the start signal, the sending of bytes can resume at any time. But after a stop signal has been sent by the terminal adaptor, the sending of bytes remains regular, although at most N_2 bytes may be sent. This regularity is guaranteed by also demanding a 'backward periodicity' of the input since the reception of the first byte after a start signal:

7. $G\ \forall b\ [in(b) \rightarrow (((\neg\ \exists b'\ in(b'))\ S_{=1/512}\ \exists b'\ in(b'))\ \vee\ ((\neg\ \exists b'\ in(b'))\ S\ start))]$.

The following axiom specifies that at most N_2 bytes may be received after a stop signal:

8. $G\ [((\neg\ start)\ S_{>N_2/512}\ stop) \rightarrow \neg\ \exists b\ in(b)]$.

Finally, we have to specify the generation of the stop and start signals. For simplicity we assume that N_1 is divisible by 5 and we define the following abbreviations to indicate the situations where the buffer is more than 80% full, respectively more than 80% empty:

$$almost\ full \equiv \exists b_1 \cdots \exists b_{\frac{4}{5}N_1+1} [\bigwedge_{\substack{i,j=1\\ i<j}}^{\frac{4}{5}N_1+1} b_i \neq b_j\ \wedge\ \bigwedge_{i=1}^{\frac{4}{5}N_1+1} buffered(b_i)],$$

$$almost\ empty \equiv \neg\ \exists b_1 \cdots \exists b_{\frac{1}{5}N_1} [\bigwedge_{\substack{i,j=1\\ i<j}}^{\frac{1}{5}N_1} b_i \neq b_j\ \wedge\ \bigwedge_{i=1}^{\frac{1}{5}N_1} buffered(b_i)].$$

Note that for each value of the constant N_1 these abbreviations can be written out to fixed length formulas. What remains is to express that the stop and start signals should be generated the *first* moment that the buffer becomes (again) almost full, respectively almost empty. In general, the first moment that a formula f *becomes* true after having been false before can be expressed for dense time domains by the formula

$f\ \wedge\ (Pf\ \rightarrow\ (\neg f)\ \tilde{S}\ f)$

where $\tilde{S}$ does not include the present:

$f_1\ \tilde{S}\ f_2 :=\ \exists t\ (t>0\ \wedge\ f_1\ S_{=t}\ f_2)$.

When one also wants to cover the case of discrete time models, the possibility of f being true the previous moment should be excluded. So, define the operator J as

$Jf\ :=\ f\ \wedge\ (Pf\ \rightarrow\ (\neg((\neg \mathbf{true})\ \tilde{S}\ f)\ \wedge\ (\neg f)\ \tilde{S}\ f))$.

Using this operator our last two axioms are:

9. $G\ [\ J\ \textit{almost full} \longleftrightarrow \textit{stop}\ ]$
10. $G\ [\ J\ \textit{almost empty} \longleftrightarrow \textit{start}\ ]$.

In fact, as can be seen from these last two axioms, the stop and start signals are not essentially needed and hence the terminal adaptor can be specified in a more abstract way only in terms of *in* and *out*. This can simply be done by substituting the equivalences of axioms 9 and 10 at the appropriate places in axioms 6, 7 and 8.

Acknowledgements

We thank those present at the DESCARTES real-time paradigms meeting in Eindhoven and the EUT group meeting in Mook for stimulating interaction.

References

[Ada] *The programming language Ada. Reference manual,*
Lecture Notes in Computer Science 155, Springer 1983.

[BKP] H.Barringer, R.Kuiper, A.Pnueli, *Now You May Compose Temporal Logic Specifications,*
16th ACM Symposium on Theory of Computing, pp. 51-63, 1984.

[CHILL] *CHILL Recommendation Z.200 (CHILL Language Definition),*
C.C.I.T.T. Study Group XI, 1980.

[K] R.Koymans, *Specifying Message Passing Systems Requires Extending Temporal Logic,*
6th ACM Symposium on Principles of Distributed Computing, August 1987.

[L1] L.Lamport, *What good is Temporal Logic?*
Proceedings of IFIP83, pp. 657-668, North Holland 1983.

[L2] L.Lamport, *Specifying Concurrent Program Modules,*
ACM Transactions on Programming Languages and Systems, Vol. 5, No. 2, pp. 190-222, April 1983.

[LPZ] O.Lichtenstein, A.Pnueli, L.Zuck, *The Glory of The Past,*
Logics of Programs, Brooklyn, June 1985,
Lecture Notes in Computer Science 193, pp. 196-218, Springer 1985.

[P1] A.Pnueli, *The Temporal Logic of Programs,*
18th Annual Symposium on Foundations of Computer Science, pp. 46-57, IEEE 1977.

[P2] A.Pnueli, *Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends,*
Lecture Notes in Computer Science 224, pp. 510-584, Springer 1986.

[W] P.Wolper, *Expressing Interesting Properties of Programs in Propositional Temporal Logic,*
13th ACM Symposium on Principles of Programming Languages, pp. 184-193, 1986.

Project No. 1277

IMPLEMENTING THE PCTE USER INTERFACE ON SUN WORKSTATIONS

Author:

Brian Hayselden, Software Sciences Limited, London and Manchester House, Park Green, Macclesfield, Cheshire, England, SK11 6SR

This paper presents a summary of the experience gained from implementing the PCTE User Interface on Sun Workstations whilst retaining the ability to run native window based tools in parallel with tools using PCTE windows. The paper assumes some knowledge of the PCTE User Interface specification.

1. INTRODUCTION

The development work described in this paper was undertaken as part of the ESPRIT Sapphire project [1], which has amongst its tasks the porting of an implementation of PCTE (Emeraude) to various systems (Sun, VaxStation 2000, VAX, HP9000 and IBM PC/AT).

At the start of the project (September '86), only a prototype implementation of a subset of the PCTE User Interface was available. This subset although fairly portable assumed total control of the display surface and therefore precluded parallel running of host window tools.

The basic objectives of the User Interface component of the port were to produce a complete industrial quality implementation (not a prototype), to report on the PCTE specification and porting problems, and to develop a validation suite for the User Interface. These basic objectives were extended to include a requirement to support parallel running of native Sun tools. In addition the ability to copy text between windows running under the two regimes was to be provided.

These extended requirements, coupled with limited prototype capability and unavailability of the prototype source, lead us to embark on a complete implementation of the User Interface using Sun software facilities where possible.

The sections below record the points of interest arising from the design and implementation of the User Interface on Sun Workstations*, followed by a review of the current state of the User Interface component of the Sapphire project including future developments.

2. TECHNICAL OVERVIEW

2.1. Software Environment

Support for PCTE Basic Mechanisms (OMS etc) is provided by extension of the Unix* Kernel. In the Sun version these changes are based on version 3.2 of the Sun Operating System which provides a high degree of System V / BSD 4.2 compatibility.

Sun Workstations* - Sun Workstations is a trademark of Sun Microsystems
UNIX* - Unix is a trademark of Bell Laboratories.

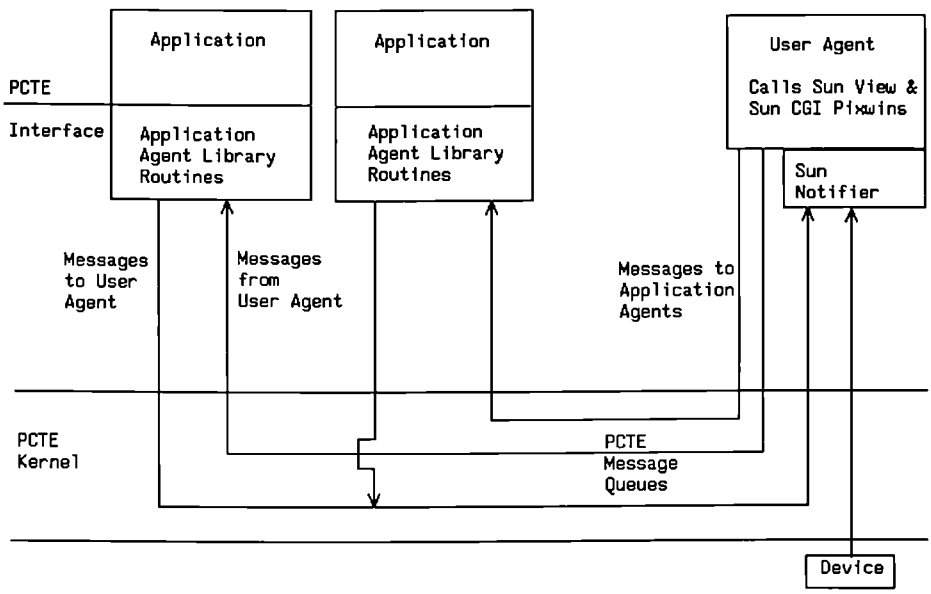
The User Interface is implemented in C, uses the PCTE OMS for storage of icons etc., and uses the Sun View facilities available with the 3.2 release. The resulting version runs on both Sun 2 and Sun 3 workstations.

2.2. ARCHITECTURAL ASPECTS

2.2.1. Process Architecture

To a certain extent the process architecture was pre-defined since the User Interface specification describes the use of a separate server (User Agent - UA) handling device interactions and application resident routines (Application Agent - AA) which interact with the UA in response to tool requests.

Overview of Process Architecture and Communication Mechanisms



2.2.2. AA <-> UA Communications

PCTE messages are used for AA to UA communication. There is a single message queue which all AA's use to send messages to the UA, and one message queue per AA used for UA to AA messages.

2.2.3. UA Event Handling

The UA has three types of asynchronous event to handle:

- events from the device;
- messages from AA's;
- an alarm handler used to handle timeout of blocking input calls and also to assist in monitoring AA's failure to respond.

The UA uses the Sun Notifier [3] to assist in managing these events. The Notifier handles all polling of the device, simplifies the multiplexing within the UA, and also allows handlers to be registered with the notifier for invocation when specific events occur.

2.2.4. Use of Sun Software Facilities

In addition to use of the Notifier mentioned above, Sun View facilities are used to support many of the User Interface objects (e.g. windows, icons, fonts), as well as to support the User Interface Bitmap and Text drawing primitives. Sun CGI (in Pixwins) is used to support the graphics functionality.

The specification supports the creation and deletion of windows etc. in a random order as well as the display and overlap of all data types (text, bitmap and graphics) within a single window. For these reasons use of the Sun facilities in an integrated manner to support the PCTE functionality is not as straightforward as might be envisaged.

2.2.5. Input Handling

All input is first seen by the Sun Notifier which directs it to the appropriate window handler. For windows owned by the User Interface this is a UA routine. This routine interprets the event if necessary and notifies the occurrence (if not ignored) to the appropriate AA where it is queued for notification to the application either by direct call of tool function (event mode) or in response to a specific input call (collect mode).

The queuing of input events in the AA avoids problems with requeuing of input notifications in transit when input modes change, and avoids problems in the UA if a rogue application is not reading its input.

2.2.6. User Interface Object Ownership

We have taken the stance that User Interface objects (e.g. windows, frames, graphics elements) are owned by the process which created them. In most cases the objects are not usable by other processes, even child processes.

Some User Interface objects are readable by more than one process:

Grey Scale Bitmap Frames;
Fonts;
Information accessed via a selection.

2.2.7. End-User Image

The end-user image was not fully defined by the prototype and aspects of the prototype image were not extensible to a full implementation. We have produced a definition of the end-user image but do not preclude changes from the style we have adopted, which has much in common with the Sun style, to a standard (to be defined) PCTE style. One aspect we expect to change is scroll bar operation.

2.2.8. Size of Development

The AA is approximately 30,000 source lines (including build state control red tape, comments and blank lines) in 150 source files and gives a total executable size of approximately 120Kb.

The UA is approximately 70,000 source lines (same basis) in 245 files and gives an executable size of approximately 900Kb. This latter figure includes the Sun Library software.

3. WHAT DID WE LEARN FROM THE DEVELOPMENT

The items listed below are in no particular order, and some of them state the obvious, but they are included since they relate to the history of the development.

3.1. Advantages of Using Proprietary Software

Use of the Sun software reduced the amount of code to be produced. For example we were able to use the available Sun functions for drawing graphics elements such as closed elliptic arcs with wide borders and fill patterns.

Less new software means less errors and hopefully less test time.

3.2. Disadvantages of Using Proprietary Software

Code size is increased since proprietary code provides functionality additional to requirements.

The code may be less portable, although use of higher level functions may ease portability.

Although there may be less errors overall, any errors in proprietary software take longer to pin down and may not be fixable or circumventable until a future release.

If you are using the host software in uncommon ways (e.g. developing a window manager on top of a window manager as we were), then the risk of hitting limits and host software interaction problems not envisaged by its designers is increased, and documentation is less likely to give directly applicable guidance.

3.3. Consistent Standards in the PCTE UI Specification

The specification manifests a lack of consistency in aspects such as function and parameter naming, parameter description and usage.

Good standards consistently applied make a specification easier to assimilate and also easier to criticise constructively. Poor or inconsistently applied standards have the reverse effect.

3.4. Specification Interpretation

As I discuss in more detail in section 4 we have raised a large number of comments on the specification and, in order to complete the design, assumptions were made in response to those comments. As one might expect the process was iterative since many areas interact and as design progressed new problems caused us to re-think our solutions.

Even where problems were identified there was frequently more than one valid solution and it remains to be seen how good we have been at choosing the right one.

3.5. Implementation Helps to Flush Out Problems

Many of the difficulties we encountered in interpreting the specification were highlighted only when doing low level design and implementation. Hopefully having produced a complete implementation a definitive User Interface specification can now be produced.

I have not yet seen the comments produced by the ESPRIT VIP project and cannot therefore judge whether we have a large proportion of comments

in common, or whether the different objectives and viewpoints have produced a significantly different set of comments.

3.6. Functional Level of the Specification

As well as highlighting problems with the specification, in terms of whether it is unambiguous and implementable, we also have some questions as to the level of facilities provided.

In some cases the need to include or exclude a facility might be decided on the basis of whether or not it is needed for a software tools environment. In other cases there is clearly a need for a particular capability, but it is not clear whether the functionality should appear directly in the interface, or be left to a higher level to provide using the existing primitives.

Examples I would cite are:

a) There is no access to the final pixel image of a window which might be constituted from several viewports.

b) For text frames: should the control of wraparound of text at line end appear as a feature of the interface or be left to tools to manage?

3.7. Advantages of the PCTE User Interface

PCTE provides a single User Interface specification document which although it needs improvement is nevertheless easily navigated and does not need extensive examples of function usage. The set of functions is fairly easy to assimilate and use.

The application view of data is separated from the end-user's view.

The provision of viewport hierarchies gives powerful capabilities.

Although different functions are provided for the handling of the different data types (Text, Bitmaps and Graphics), the display mechanisms (viewports and windows) are type independent and imply no restrictions on combining data types within windows.

PCTE provides a higher facility level (Menus, Window Titles, Window Scroll Bars etc) than X Windows.

The adoption of a server architecture leads to smaller application code size.

3.8. Disadvantages of the PCTE User Interface

PCTE limits access to some primitive operations (e.g. control over line joining algorithms). In some cases this may actually be an advantage for a software tools interface.

PCTE has some missing functionality (e.g. there is no capability for accessing the pixel image of a window or text or graphic viewport). It remains to be seen which 'missing' facilities are important for the target usage. The functionality can always be added.

The performance is likely to be slower than that of a tool using native facilities directly since an extra process switch is involved. This is evident when drawing mouse trails for example, but the overheads can be limited. It remains to be seen what the requirements are in a software tools environment for facilities of this nature where direct

end-user feedback is required and the amount of tool activity in response to a single event is small.

There is no doubt that the Functional Specification needs improvement. This is being addressed.

There is a shortage of tools using the interface. The PACT toolset will of course address this problem.

3.9. Interworking with the Host System

As stated in the introduction we wished to provide the capability of running existing Sun tools in parallel with tools using PCTE windows. This gave us a number of benefits and also some difficulties.

It is clearly commercially beneficial to allow concurrent running on systems with an established customer base.

We were able to use the window based DBXTOOL during testing.

TTY emulation was provided by the native Sun facilities. In the future it is expected that a standard method of mapping TTY onto a PCTE window will be developed.

We used the Sun facilities for icon image generation etc. pending the availability of PACT tools.

Direct visible comparison of the performance of simple tests was possible (e.g. rubber banding and mouse trailing).

At first sight Sun selection handling and PCTE selection handling appear to have a lot in common and time was spent trying to evolve a unified selection system. However there are fundamental differences within the default schemes such as PCTE's inclusion of multiple selections and different data types and Sun's use of multiple key operations to give composite actions.

These differences lead us to adopt a narrow interface between the two schemes whereby text could be copied from a Sun window to a PCTE window and vice-versa using the Sun PUT and GET operations.

3.10. Don't Start From Here!

We started from a far from ideal position since the Functional Specification was unproven, there was no documentation of the rationale which might have resolved ambiguities and there was no full implementation to refer to for clarification.

3.11. Interaction between End-User Image and Application

Although separation of the application interface from end-user image is generally a good thing, it is nevertheless essential to identify the end-user functionality since this can impact the tools view of interaction sequences.

A tool writer needs to know that the interaction style for the tool will not conflict with system function operations, and will not confuse the end-user.

4. The PCTE User Interface Specification

Throughout the development of the PCTE User Interface on Sun workstations a significant number of observations were made as to inconsistencies, contradictions, ambiguities and information missing from the specification. Although some of these observations related to facilities and capabilities, a large proportion demanded resolution before a consistent low level design could be produced.

These observations excluded any of a typographic nature except where such errors affected the technical meaning of the specification. Thus spelling errors in text descriptions were not deemed significant. Incorrect spelling in field names in structures or ambiguous text descriptions were treated as significant.

Each comment was recorded and given one of four classifications. It should be noted that apart from class [A] the class of the comment does not give any real guidance as to the extent of its impact. Some involve simple changes, others might involve changes to several functions.

[A] Typically a minor clarification such as an additional error return value or clarification of parameter value ranges. Statements are given this class if they are likely to be correct and/or if the cost of change is likely to be small (e.g. re-compile).

[B] These are comments which affect the tool interface as specified but not the basic functionality supported. Examples are details of parameter interpretation and valid sequences. No impact on end-user image is envisaged if the assumption is changed. The cost of change is typically higher than for [A].

[C] Similar to [B] but with end-user image implications. For example statments are made as to what is displayed in boundary conditions for graphics elements such as circles with zero radius.

[Z] Comments highlighting an aspect which we think could benefit from change. For these comments the Sun implementation follows the PCTE specification, but the comment registers our reservations as to the usefulness of a facility which is provided and/or its method of presentation, or our view that a missing facility should be provided.

At the time of writing (July '87) the numbers of comments raised were as follows:

[A]	[B]	[C]	[Z]
127	147	99	59

The SAPPHIRE project is reviewing these comments and proposals will be placed before the PCTE Interface Management Board in due course. It is hoped that a significant proportion of our assumptions (classes [A], [B] and [C]) will be accepted as per our implementation.

4.1. Further Breakdown of Class [Z] Comments

The [Z] class includes comments on the need to review aspects which do not affect functionality such as function naming standards as well as comments on possible additional functionality such as access to final window image. A more detailed analysis of these gives:

Comments relating to consistency and clarity of specification 27
(E.g. The need for naming standards)

Comments questioning the functionality 32
(E.g. Lack of a BATCHING facility and access
to window pixel image)

5. CURRENT STATE AND FUTURE DEVELOPMENTS

5.1. Status of "Sun View" Version

The User Interface implementation on Sun is available for evaluation. At the time of writing (July '87) we have a version which is complete except for text frame output. A complete version is planned to be available in October '87.

The July version has not been fully tested, and we are developing a PCTE User Interface validation suite in parallel with completing the implementation.

5.2. GIE Emeraude Sun Version

A version evolved from the initial prototype is being developed by GIE Emeraude. This version will run on Sun, but runs at the PIXRECT level and so precludes concurrent use of Sun Window tools.

5.3. X Windows Version

The SAPPHIRE project is now actively looking at producing a version of the User Interface running on X Windows Version 11. Such a version is needed for the VaxStation and the HP9000. Since Sun now intends to support X Windows we may choose to make this version common.

X Windows provides an interface to support window managers and provides most of the facilities needed to support the PCTE User Interface. The only significant problem is that the PCTE User Interface allows event driven input whereas X Windows provides on demand reading only. The solution to this problem seems to require the involvement of a separate process (the UA) in the input path in these circumstances.

5.4. Functional Specification Update

It is important that the functional specification is upgraded to address outstanding comments. Amongst the many problems in resolving these comments, resolution of comments relating to additional and redundant functionality demands the involvement of potential users of the interface (e.g. PACT and SAPPHIRE tool writers).

6. IMPACT TO DATE

An early (April '87) version was demonstrated at the Ada Europe Conference in Stockholm. This demonstration showed PCTE as a whole, using the User Interface facilities to display information obtained from the OMS and to direct navigation through the OMS.

6.1. Evaluation

A version of PCTE User Interface is scheduled for delivery to ESTEC for evaluation, and some ALVEY projects will take versions.

Another aspect of Project SAPPHIRE is to evaluate PCTE by porting

ECLIPSE to PCTE. The database aspects of ECLIPSE already use PCTE, and the remaining work involves mapping the ECLIPSE application interface (which currently uses Sun window facilities directly) onto the PCTE User Interface.

6.2. Summary

The Sun implementation works and is available despite all the difficulties encountered. Further work is needed and is in hand to conclude the development.

7. REFERENCES

[1] Project Sapphire 1229 (1277) PCTE Portability Annex 1. CAP Industries Limited.

[2] Sun PCTE Porting Experience Report. Software Sciences Limited.

[3] Sun View Programmer's Guide. Sun Microsystems. Revision A of 17 February 1986.

Project No. 1277

THE SAPPHIRE PROJECT : BUILDING CONFIDENCE IN PCTE

Mike Tedd

Department of Computer Science, University College of Wales, ABERYSTWYTH, Wales, UK

1. INTRODUCTION

A new generation of software tool support interfaces has been defined. In Europe there is the Portable Common Tool Environment (PCTE) [1]; in the US there is the Common APSE Interface Set (CAIS) [2], soon to be succeeded by CAIS-A. References [3] and [4] give more background and some comparison of these interfaces.

All these tool support interfaces attempt to provide a much more powerful environment for the software tool than those provided by Unix and typical manufacturers' operating systems. In particular, they provide sophisticated facilities for structuring and manipulating data, and for ensuring the integrity of the data. Also, by not being proprietary to any computer manufacturer, and hopefully becoming widely available, the existence of PCTE gives the promise that tools using it will be portable across a wide range of computer hardware.

PCTE is one of the notable successes of the ESPRIT programme. It is much more powerful than CAIS. A production quality implementation of PCTE, Emeraude, is already on the market, whereas quality implementations of CAIS have only just started to be worked on. The CAIS-A design has used PCTE as a prime source of ideas, so it should be comparable in sophistication, but it is years behind PCTE in development.

PCTE represents a real opportunity for the European Software Industry. Its widespread use in Europe could mean much expanded markets for software tools, with consequent incentives for European tool-writers to invest, and eventual benefits to European software developers when they reap the gains from the availability of better tools.

However, recognising the virtues of PCTE, and making sure that it is widely used in Europe, are two different things. There is natural caution with the introduction of any new technology. No one likes to be among the first users who may meet problems; people like to see that the benefits of sophisticated features are real; there is

The Sapphire project is part funded by the Commission of the European Communities under the ESPRIT programme, project 1229(1277). The participants in the project are CAP Industry Ltd. (UK, prime contractor), Software Sciences Ltd. (UK), GIE Emeraude (France), and UCW Aberystwyth (UK).

concern that much sophistication might have performance penalties; a range of general tools needs to be available; the interface needs to be widely available to gain the portability benefits.

Bridging this confidence gap is what the Sapphire project is all about. The project has three main areas:

- Making PCTE widely available
- Gaining experience of using PCTE
- Studying the performance of systems built on PCTE

2. MAKING PCTE WIDELY AVAILABLE

2.1. Overview

The Emeraude production-quality implementation of PCTE was developed on the Bull SPS-7, which is not much used outside France. The Sapphire project is porting Emeraude to the following widely available systems:

SUN 2 and 3
IBM PC/AT
DEC VAX
DEC VAX Station 2000
HP 9000 series 300

The first major port, to the SUN, has already been demonstrated.

An initial study predicted, and the project's experience has confirmed, that porting Emeraude is reasonably straightforward if the target architecture already has an implementation of Unix System V or similar. The exception to this is the area of User Interface, where PCTE's rich facilities must be built on the very different facilities and concepts of different architectures. Our approach to this has been first to implement the user interface for the SUN, using SUNView as a base, and then to base our implementation on X Windows Version 11, giving a much more portable implementation for the future. The companion paper [5] describes this strategy in detail.

An important part of a successful port is to validate its functionality. A PCTE Validation suite is being developed, and one public deliverable of the project will be a report describing the Suite and documenting the results of its operation on each of the five ports.

The benefits of this part of the project will not just be the ports themselves. By demonstrating that PCTE can be implemented on such a range of architectures, by reporting on our experiences, and by improving the portability of Emeraude itself, we will contribute to confidence that PCTE can be made available on many other

architectures within a reasonable budget and timescale.

2.2. The SUN Port

The SUN port was first demonstrated at the Ada Europe conference in Stockholm, in May 1987, and can be seen at the exhibition associated with this ESPRIT Technical Week. There will be several releases of this port during 1987.

The host configuration is a SUN 2 or SUN 3, running SUN 3.2 OS, with 4MB of main memory, and at least 71 MB of disc; TCP/IP protocols are used for distribution. Later versions are expected to use version 4 of the operating system, and eventually to move to OSI protocols for distribution.

2.3. The VAX Ports

The first VAX Port should be available late in 1987. This will be hosted on Ultrix 2.0, and will have full PCTE functionality except for the user interface, which will be a 'teletype subset' (since typical VAX systems lack bit-mapped screens).

During 1988, a full version of PCTE will be available on the VAX Station 2000. This will have a complete user interface, based on X Windows.

2.4. The PC/AT Port

The IBM PC/AT, its clones, and its successors (the PS/2, and other 80386 based systems) are the most widely available systems capable of running PCTE. So it is very important that PCTE be ported to this architecture.

The port is based on Microport's version of Unix System V. The first release will be early in 1988. It requires a PC/AT or similar, with 2 MB of main memory, EGA graphics, and a mouse.

2.5. The HP 9000 Port

The final port to be undertaken is to the Hewlett-Packard 9000 Series 300. This is a 68020 based system, with a high resolution display, keyboard and mouse, and Ethernet networking. The port will be based on the host's version of Unix, HP-UX 5.2. It is planned to complete this port during 1988.

3. GAINING EXPERIENCE WITH PCTE

3.1. Overview

This part of the project will move some very substantial toolsets onto PCTE, and report on the problems met and the benefits perceived. The toolsets are Eclipse, a major Integrated Project Support Environment (IPSE); Fortune, a sophisticated documentation support system; and an Ada compiler. The relationship of PCTE and CAIS is also being studied.

3.2. Eclipse

This is one of the major IPSE developments that are part of the Alvey research programme in the UK. Eclipse provides three major toolsets to the end user, to support the LSDM methodology, to support the MASCOT methodology, and to develop Ada programs. These toolsets are themselves very dependent on generic Eclipse tools, notably a generic design editor. All the tools of an installation present a uniform interface style to the user, and make use of the Eclipse database, which provides uniform access to fine-structured data as well as coarse data.

Eclipse Version 1 was hosted on Unix, using SDS-2 and Foundation (two Software Sciences products) to support the coarse level database.

Eclipse Version 2 is hosted on PCTE, using its Object Management System (OMS) to support the coarse level database. The total dependence of Version 2 on PCTE illustrates the confidence felt in PCTE.

After a comprehensive study had been made of the problems of moving to PCTE, implementation has proceeded quickly; all three toolsets were first demonstrated running on PCTE at the Alvey conference in Manchester in July, and can also be seen at the exhibition here.

3.3. Fortune

This is another substantial Alvey project. The project will produce a documentation system for software engineers, running on a network, with individual workstations accessing a common database. Various types of component (e.g. diagrams, code, text) and the cross-referencing between them are supported. The advances over existing systems lie in Fortune's configurability for different methods, and ability to interact with other tools in the software engineering environment.

The actual rehost of Fortune lies outside the Sapphire project, but one report of Sapphire will be a study of the problems and benefits of moving Fortune to PCTE. An obvious immediate benefit is that the PCTE version is much more portable than the native SUN version.

3.4. Ada Compiler

CAP have already retargeted the Telegen Ada compiler to the 80286 architecture. As part of Sapphire, this compiler will be retargeted, and rehosted, to the PC/AT running PCTE. This includes the integration of the compilation system with PCTE so that the OMS of PCTE is used to support the Ada program library.

Again we will assess the benefits, and any problems, in using PCTE for this system.

3.5. Relationship to CAIS

This area of the project will study the relationship of PCTE and the US standard, CAIS. A mapping of CAIS onto PCTE will be designed and implemented, and comparisons made between PCTE itself, CAIS implemented on Unix, and CAIS implemented on PCTE. By attendance at KIT meetings, the definition of CAIS-A is being followed.

4. STUDYING PERFORMANCE

Both for Eclipse and for Fortune, it will be possible to conduct trials comparing performance of the PCTE-based versions with the non-PCTE-based versions of the software. For Eclipse, the comparison will be between Version 1, based on SDS-2 and Foundation, and Version 2, based on PCTE. For Fortune, the comparison will be between the native SUN version and the PCTE-based version. This experience will be captured in project reports.

REFERENCES

[1] PCTE. A Basis for a Portable Common Tool Environment. Functional Specifications. Fourth edition. 1986.

[2] Common Ada Programming Support Environment (APSE) Interface Set (CAIS). DOD-STD-1838, 9 October 1986.

[3] Lyons T.G. & Tedd M.D. Recent Developments in Tool Support Interfaces, CAIS and PCTE. Ada UK Conference, York, January 1987.

[4] Lyons T.G. & Tedd M.D. Technical Overview of PCTE and CAIS. Ada UK Conference, York, January 1987.

[5] Hayselden B. Implementing the PCTE User Interface on SUN Workstations. ESPRIT Technical Week, 1987.

Project No. 974

A KNOWLEDGE BASED ENVIRONMENT FOR S/W SYSTEM CONFIGURATION REUSING COMPONENTS

J.-F. Cloarec (*), R. Valent (**)

The paper presents a concise overview of the KNOSOS project (number 1221) and gives the results obtained in the first year of its life. During this period, a mixed strategy was used :

- According to the "waterfall" model of life cycle, the Users' needs and Users' requirements (URD) were investigated, giving the view of major industrial partners from Telecommunications and Space applications.
- From the state of the art on Software Reusability, KNOSOS approach and expected support were defined.
- Employing rapid prototyping, typical examples of components reuse in large programs are being implemented. Their experimentation, now in progress, will produce a refined version of the URD and give early inputs to the Software Requirements Document (SRD) of the precompetitive prototype.

Necessary features of the knowledge representation models and of composition algorithms were in this way identified. KNOSOS is classified as an intelligent environment to support the "product view" of reusability techniques (Esprit work programme area 2.3.3.5).

1. INTRODUCTION

The KNOSOS objectives are the development of knowledge rich environments and active methods to support the software engineering.

A precompetitive prototype shall demonstrate that reusing components to build large to very large industrial software, is a real goal to achieve in the next future.

The KNOSOS team approach is that of the industrial S/W life cycle, this means that the development is preceded by extended work to define the Users' needs [1] and to synthetize common Users' Requirements Definition (URD) [2].

In order to shorten the time needed by standard procedures using the so-called "Waterfall" model, intensive usage of rapid prototyping is planned.

In the first part of the paper are presented the KNOSOS team's main findings in Users' needs and requirements, the second part outlines the KNOSOS software

(*) National Telecommunication Research Center (CNET), LAA/SLC/AIA, Route de Trégastel, 22301 Lannion Cedex, France.

(**) Engineering Software International (ESI), 20 rue de Saarinen, SILIC 270, 94578 Rungis Cedex, France.

reusability approach, the third describes the first results concerning component description and handling, deduced from preliminary partner requirements and prototype experiments.

2. KNOSOS USERS' NEEDS AND REQUIREMENTS

2.1. Users' needs

2.1.1. The State Of The Art In Reusability Within The Consortium

The KNOSOS Consortium is composed of three "hi-tech" companies in the fields of Telecommunications, Electronics and Space, a Research center, and two Software houses. Their experience and needs in software reusing, is various and different. This is maybe the main asset of the project, because a large spectrum of real problems could be identified and investigated.

The state of the art within the Consortium is mainly characterized by the following points :

- Methods and Tools consist mainly of libraries of components, but no specific reusability tools are presently used. What is reused is mainly the design, through human experience and training.

- Reusability ratio is :

    . high within products/projects,
    . low to medium, between products/projects.

- Critical issues are variable with respect to the industry, but the following are identified as target of reusability :

    . Specification,
    . Documentation,
    . Design,
    . Test,
    . Maintenance.

2.1.2. What Is Expected From KNOSOS

KNOSOS shall demonstrate that it can bring real support to obtain :

- Increase in productivity,

- Project management enhancement in terms of :

    . Estimation of cost,
    . Time scheduling,
    . Resource optimizing.

- Quality assurance improvements.

2.2. Users requirements

Some requirements are considered essential and others only desirable. Priority order is to be defined later in the revised URD, taking into account the results of the feasability study.

2.2.1. General Requirements Are Expressed In Terms Of :

- Software engineering support for :

  . Whole life cycle,
  . Various types of software, eg. real time ...,
  . Evolution,
  . Customization.

- Compatibility with existing software development tools and environments,

- Portability,

- Distributed environments,

- Standards enforcement,

- Quality control procedures retrieval,

- Reuse methodology :

  . How to use KNOSOS,
  . How to structure components, eg. a specification for headers,
  . How to reclaim the existing components.

2.2.2. Functional Requirements Are Expressed In Terms Of :

. Development support,

  For top down approach, configuration management, support to prepare an offer, component history, interface with existing development and management tools ...,

. Software configuration and construction,

  With list of unsatisfied requirements, components which partly satisfy requirements, rules to combine components, control of the assembling criteria such as : run time, memory size ...,

. Software configuration management,

  Taking into account several specification levels, automatic retrieval of information by scanning component headers, dependence links between abstraction levels, transformation methods from levels to levels, richer functional role description, customized management and version control, troubles reporting, changes control, history management, trust vector ...

2.2.3. Requirements On Man/machine Interface

. Supported by IBM PC-like and SUN-like workstations,

. Windows and mouse,

. Keyboard/ASCII-terminals,

. Interactive and batch,

. Compatibility with PCTE users' interface,

. Display with graphic features.

2.2.4. Operational Environment Requirements

Concern the operational system on which KNOSOS should be installed, ie. UNIX, VM, VMS.

2.2.5. The Feasibility Confirmation Will Be Assessed On The Following Studies :

. Representation of specification,

. Levels of abstraction,

. Transformation method,

. Functional-role description,

. Automatic extraction of information for headers.

This is to be validated with prototyping of pragmatic Users' examples.

3. THE KNOSOS SOFTWARE REUSABILITY APPROACH

3.1. Current Software Reusability Approaches

Software reusability is becoming a great concern as well for industrial software producers, as it has been developed in the previous chapter, than for research labs. This interest has been emphasized during the latest Software Engineering Conferences where Software Reusability Tutorials were proposed and witnessed a wide attendance [3], [4].

We can divide software reusability approaches into two basic groups [5] depending on the nature of what is to be reused : reusable building blocks, and reusable patterns of analysis or design. We will discuss also the main issues on software reusability.

3.1.1. Reusable Building Blocks

The components been reused are atomic building blocks such as subroutines, functions, program modules ... which are organized and combined according to well-defined rules. The emphasis is on the development, the accumulation and the reuse of program components themselves, stored in an application library [6], or on the definition of organization and composition principles, for example pipe mechanisms in UNIX that allow and encourage the reuse of programs [7], or the object oriented programming that is able to produce new object classes by specialization, using inheritance mechanisms : versions of procedures that require modifications are added to a specialized object class, while the unchanged procedures are inherited [8] [9].

3.1.2. Reusable Patterns Of Analysis Or Design

In this approach, more than reuse of passive components, reuse is the reactivation of the generation mechanisms to produce new software systems or components.

Language-based Systems emphasize on the notation used to describe the target system. The languages used are Very High Level Languages (VHLLs) or Problem Oriented Languages (POLs). The firsts rely upon a small number of semantically neutral primitive constructs such as mathematical sets, abstract types or predicates [10], while the seconds incorporate a rich set of problem domain specific information.

Application Generator Systems, like DRACO [11], rely on a general knowledge and expertise about an application domain and their use to produce new software systems like new solutions to problems in this domain.

Transformation Systems allow the system designer to specify the target system in a VHLL, and to perform incremental changes, from a terse but poorly performing specification, into an efficient executable form. For example systems to transform LISP programs into FORTRAN code [12], or transformation systems used as a methodology for program development [13].

3.1.3. Issues On Reusability In Programming

While the reuse of building blocks is the easiest to implement and can give immediate results, the reuse of patterns of analysis or design is thought to be the more promising for the long haul. When software evolution is the primary requirement, the record of the main design decisions (Design Plan), like for hardware Redesign [14], is essential.

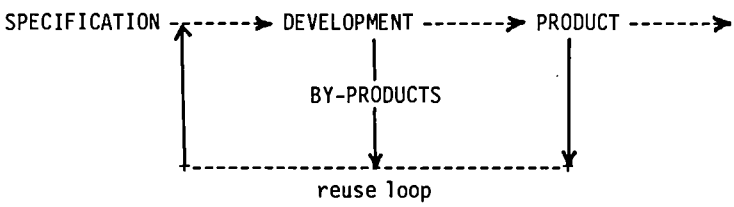
The main issues are the economic payoff of reusability compared to the investment, this obviously depends widely on the software domain of application, and the impact of reusable programming on software models that designers must bear in mind [15].

3.2. KNOSOS Approach

KNOSOS intends to focus on "Programming in the Large", that is to say reuse software by constructing and managing complex systems made of a collection of components. This is opposed to "Programming in the Small", that is producing new software components from control blocks and programming idioms. We think that the two approaches are complementary but that the first one is more tractable and promising for the short and mean terms, as we can expect to devise methods and tools independently of any programming languages and specific machines. The reuse of components seems to be both more adapted to the industrial needs, and more likely to produce interesting results, analogous to the large development, use and reuse of hardware components.

Our assumption is that most of the two software reusability approaches, reuse of building blocks and reuse of design analysis, can be captured using reuse of software components of different types and at different abstraction levels (software development phases).

The general reuse process is a follow :



The development of software yields :

- a main product, that is a software system made of a collection of modules and the corresponding documentation,

- and by-products that are intermediate descriptions, test procedure and data, and use of tools.

Reusability is then the reuse of a library of products and of a collection of by-products, and we believe that both can be considered as software components of two kinds : simple components and complex components (packages and sub-systems).

Simple components are parts of software that :

- are identified as management items,

- fulfill a functional role,

- have interconnection interface,

- are able to be assembled to form complex components,

- can be transformed using tools.

Most of the time, these components are described or implemented using semi-formal or formal languages like specification languages, VHLLs, POLs, or programming languages. So we can expect them to have fairly well defined features, and that facilities to help extract information relevant to reusability could be developed in the near future.

They are defined at several levels of abstraction corresponding to the several stages of software production that are roughly :

- software specification,

- design,

- implementation,

- test and integration,

- maintenance and evolution,

- documentation (in parallel with every stage).

The application of human skill or production tools like generators, transformations, compilers, configurations ... transforms or refines a level to another. For example an executable component (module) is produced from a source component using a compiler.

These tools are to be described as components too.

Taking into account several abstraction levels can dramatically improve potential reusability like in [16], because the higher levels can be devised to be environment and/or implementation independent.

4. KNOSOS FIRST RESULTS

4.1. Principles And Functionalities

To meet the users' requirements, the aim is to propose a sheme, and corresponding tools, to set a comprehensive library of components. On the one hand, we intend to be able to reuse them, on the other, we want to use experience to elaborate and express reusability principles so as to improve reusability of components (Expert System Approach).

For this, the main KNOSOS basic functionalities are :

- Description of components and expression of knowledge about them,

  We are defining a *model of components* that is to be the basis for the formalization of one or several description headers.

  This model must take into account the modularity principles of "Information Hiding" : there is no needs to know its implementation to use a component, "High Cohesion" : each component is made of strongly related sub-parts, and "Low Coupling" : every component can enter in several combinations.

  The implementation will use a Knowledge Representation System based on the Frames Model 17 and on an Object Oriented Language. Basic simple components are to be prototype descriptions of classes, consistency and reusability principles are to be procedural attachments and rules sets.

  The main items of this model, as we can see it to-day, are :

  . Identification,

    With name, version, revision, type (source, executable, documentation, specification, test, data ...)

  . Functional role,

    To describe the performed functional role, and sub-roles (to capture the functional architecture of systems) ...

  . Interface description,

    Featuring imports, exports, shared data structure ...

  . Environment,

    Taking into account operating system, machine, language, tools, quality level ...

  . Implementation,

    For algorithm description, memory occupancy, efficiency, special constraints ...

  . History.

    To keep history of tests, evolutions, reasons behind changes ...

- Component retrieval,

  One of the impediments in reusing a library of components is the difficulty to know and understand what we have. General classification schemes are incomplete and never fit well with the actual needs.

  We propose, instead of classification, to study and give *general search facilities* implementing Artificial Intelligent Algorithms like : use of partial pattern matching, unification, search for similarities ... and exploiting the Knowledge Model of components.

- Component assemblage

  The last step in reusing components is the *construction of complex systems* out of them, while controlling the consistency and quality of the composition.

  We will study and implement algorithms and interactive procedures derived from Problem Solving Algorithms used in Artificial Intelligence. Starting from the description of the target functional roles, the constraints or principles to enforce, and the evaluation criteria, components descriptions and rules are to be interpreted as knowledge chunk to combine in order to find a valid composition. According to 18 , a composition must comply with the "Well Formed Composition Rules", that is a solution must be "Complete" : every import or needed resource is satisfied, "Conflict Free" : there is no exportations or resources incompatibilities, and "Minimal" : there is no useless components.

  We intend to study and propose satisfactory algorithms to find efficiently a first acceptable solution, admissible ones to find optimal solutions regarding quality criteria, and exhaustive algorithms to find the best incomplete solutions when no exact solution is found. The goal is not to automatically compose new systems, but rather to propose partial, yet consistent, solutions, while leaving delicate decisions to the designer.

4.2. Types Of Knowledge To Be Represented

The Users needs and the corresponding prototype experiments, lead us to characterize the different types of knowledge to be represented in KNOSOS.

4.2.1. Model Of Software Components

The main types of components are :

- Code Modules,

  With identification, functional-role, functional architecture, assemblage interface, environment and implementation constraints ...

- Documentation,

  With document structuration, links between documents, summary of content, associative access using keywords, corresponding dictionary of concepts ...

- Test Modules,

  They are ordinary code modules but with additive dependencies between tests and executable modules, and hidden dependencies from test to test (history of tests, tests constraints and data).

- Abstract Components,

  Descriptions of specifications, algorithms ... are considered as common components as they are supposed to be described using some specific language (VHLLs or POLs). But they are likely to have also new types of relationships from one level of abstraction to the others (transforms, refinements, generators ...).

- Facilities and Tools.

  For example Compilers, Editors, Generators ... They also must be considered as common components but with specialized relationships with other types of components.

4.2.2. Rules And Know-how

These rules are to express high level relationships between components with the flexibility needed to be method and application independent, but adaptable to the preferred ones in a software factory.

We can list :

- Consistency rules,

  To check types, managerial laws (version, revision ...) and so on,

- Good Decomposition in modules,

  To enforce principles of modularity,

- Reusability Principles,

  To test criteria to define reusable components such as modularity, well defined interfaces, customization, parametrization ...,

- Good Composition of Systems,

  To be able to build "Well-formed Compositions", while taking into account quality criteria and environment constraints,

- Know-How Rules,

  To support the use of general facilities (Compilers, Editors ...) and special tools, and to give expert rules and rules of thumb to help the interactive selection of components,

- Heuristic Rules,

  Used by the retrieval and the composition algorithms to control their search for plausible components.

4.2.3. History Of Changes

It is important to be able to manage the evolution and maintenance of the different components, that is mainly :

- Describe changes,

- Document the reasons behind changes,

- Keep history of changes.

4.2.4. Complex Software Systems And Packages

Complex systems in use and delivered to customers must be represented and managed with their composition, status, customer, site ...

4.2.5. Numeric And Procedural Knowledge

The above types of knowledge are mainly symbolic ones. But we have also to deal with numeric and procedural knowledge such as : size of memory needed, execution times, heuristic functions to compare the quality of two sub-systems, criteria for reusability ...

4.3. KNOSOS Experiments And Study Strategy

From the expression of the Users' Needs, described in the Users Requirements Definition (URD), KNOSOS is being studied in three main steps :

- KNOSOS rapid prototyping and experiments,

  They use mainly the SIBEMOL [19] prototype Configurer System implemented using the Knowledge Representation System ROSACE, both systems developed at CNET.

  We study typical reuse problems proposed by the different partners, and rapidly prototype them to understand better the problems as well as the possible solutions. For this, SIBEMOL is modified on ad-hoc basis, without spending much time for integrating the resulted extensions.

  Up to now, four typical cases are being studied :

  - *Evolution of a software system* and reuse of the test programs and data (YARD example [20, 21]),
  - *Modification dependencies*, or how to find accurately the components actually affected by a change (ALCATEL example [22]),
  - *Construction of a complete system* out of components produced and manipulated by several development tools (MATRA-Espace [23] and DORNIER [24] examples),
  - *Customization of a man-machine interface* using menus, with advice (documentation) to use them (ESI example [25]).

  These examples stemmed from the partners experience with their own configuration tools like LIFESPAN (YARD) and VM/SE (ALCATEL).

  They are helping to explicit the Users' needs, to devise the KNOSOS reusability principles and functionalities, and to identify the types of Knowledge Representation and of Reasonning to perform.

  They will be developed and completed later and lead to the Revised URD.

- KNOSOS precompetitive prototype and specification (SRD),

  The second step is the specification of the main KNOSOS functionalities from the Revised URD, and the integration of solutions taking into account priority order. A first full fledged KNOSOS precompetitive prototype is to be developed to validate the general proposition, with the following features :

  - a *general component model* [26, 27] (formalized header) and rules implemented with ROSACE,
  - *search and interactive composition algorithms* implemented in Common-Lisp [28],
  - *interface with the extended Relational Data Base DAMES* (ESI) to manage the factual base (component headers, library of components),
  - *man-machine interface* using windows/mouse/keyboard systems.

  This prototype will be developed on a workstation running UNIX, that is a system compatible with the European PCTE [29] System.

- KNOSOS special instantiation and customization.

The KNOSOS Esprit project will end with the above pre-industrial prototype and the corresponding general specification. This later defines a generic tool that must be instantiated and customized for specific companies environments and goals. For example, transfer to a given machine and system, development of new man-machine interface, interface with preferred Relational Data Base, production of facilities to (semi) automatically extract description and document headers from source code (depending on programming languages), integration to existing configuration systems, extension of the component model or headers ... This is left to further users appreciation. But basically, using Artificial Intelligence techniques like Knowledge Representation and Rules sets, KNOSOS will be flexible and easily adaptable to given procedures and users.

5. CONCLUSION

The study "Users needs" has shown that :

- Reusability is a real fact in industry,

- Various ratios of reusability are now obtained, better ones within one project than between projects,

- There is a real need to set-up a reusability methodology and to provide corresponding environment and tools.

The users requirements document stressed that :

- Common requirements can be agreed upon,

- Research work shall be further done on a limited number of topics such as levels of abstraction of components, use of transformation method, refinement of functional role descriptions, and headers automatic generation,

- Prototyping of pragmatic examples is a good approach to refine the URD.

To meet the Users' requirements, the KNOSOS approach to software reusability is a "Programming in the Large" one where a comprehensive model of software components, aimed to be reusable at the code level as well as at the design level, is being elaborated and implemented using a Knowledge Representation System. Powerful algorithms, implementing Artificial Intelligence techniques, are also studied to retrieve and assemble components to help the building of complex systems out of them, while enforcing consistency, managerial and expert rules, and promoting the reuse of existing components.

We believe that the future use of such a tool, connected to an extended Relational Data Base, and provided with a friendly man-machine interface, (i) will greatly improve software reusability in industry, and (ii) will help exhibit criteria and principles to devise better reusable components.

ACKNOWLEDGEMENTS

We should like to thank all partners ALCATEL (F) : J.-P. Quillien, J.-P. Hubaux, J. Nicolas, CNET (F) : J. Collet, DORNIER (RFA) : Dr. Katzenbeisser, F. Billich, ESI (F) : J. Dubois (our project manager), J.-L. Gregis, MATRA Espace (F) : B. Meijer, J.-P. Denier, and YARD Software (UK) : I. Pirie, A. Tilbury, for their contribution to the KNOSOS project.

REFERENCES

[1] KNOSOS Team, "Users Needs Consolidated Report", KNOSOS Document 26/K/draft, 15/5/87

[2] KNOSOS Team, "URD Consilated Report", KNOSOS Document 25/K/Draft, 15/5/87

[3] Peter Freeman, "Software Reusability", Tutorial, 3rd. Software Engineering Conference, May 27 1986, Versailles, France

[4] John Goodenough, "Software Reusability", Tutorial 5, 9th. International Conference on Software Engineering, March 30, 1987 Monterey, Ca, USA

[5] Ted J. Biggerstaff and Alan Perlis, "Forewords" (Special Issue on Reusability in programming), IEEE Transactions on Software Engineering Vol 10, N° 5, Sept 1984

[6] Robert G. Lanergan and Charles A. Grasso, "Software Engineering with Reusable Designs and Code", IEEE Transactions on Software Engineering Vol 10, N° 5, sept 1984

[7] Brian W. Kernighan, "The UNIX System and Software Reusability", IEEE Transactions on Software Engineering Vol 10, N° 5, Sept 1984

[8] Gael A. Curry, Robert M. Ayers, "Experience with TRAITS in the Xerox STAR Work-station", IEEE Transactions of Software Engineering Vol 10, N° 5, Sept 1984

[9] Joseph A. Goguen et al, "Programming with Parameterized Abstract Objects", in Theory and Practice of Software Technology, 1982, pages 163-193

[10] M. Lemaitre, "Définition et Contrôle d'une Base de Données Projet pour un Système d'Assistance à la Programmation : SPRAC", ONERA-CERT/DERI n° 1/3180/DERI, 1982

[11] James M. Neighbors, "The DRACO Approach to Constructing Software form Reusable Components", the REUSE Project, University of California, Irvine. Workshop on Reusability in Programming, Newport, sept 1983

[12] James M. Boyle and Monagur N. Muralidharan, "Program Reusability through Program Transformation", IEEE Transactions on Software Engineering Vol 10, N° 5, Sept 1984

[13] T. E. Cheatham, J. A. Townley, G. H. Holloway, "A System for Program refinement", Harvard University, Cambridge (MA), TR 5-79, August 1979

[14] T. M. Mitchell, L. I. Steinberg, S. Kedar-Cabelli, V. E. Kelly, J. Shulman, T. Weinrich, "An Intelligent Aid for Circuit Redesign", AAAI-83, Washington DC, August 1983

[15] Elliot Soloway and Kate Ehrlich, "Empirical Studies of Programming Knowledge", IEEE Transactions on Software Engineering Vol 10, N° 5, Sept 1984

[16] Y. Hori, S. Kimura, H. Matsuura, "Reusable Design Methodology for Switching Software", NTT Electrical Communication Labs, Tokyo, 1985

[17] M. Minsky, "A Framework for Representing Knowledge" in The psychology of computer Vision, P. H. Winston (ED), Mc Graw Hill, 1975

[18] A. N. Habermann, D. Perry, "Well-Formed System Composition", CMU-CS-80-817, March 1980

[19] J.-F. Cloarec, J.-F. Cudelou, J. Collet, "A Configurer for Switching System Software Based on the Knowledge about the Assembling of Program Modules", IFIP Working Conference, Budapest, September 1984

[20] A. Tilbury, "YARD S/S KNOSOS Test Case", KNOSOS Document 27/Y/Rev0, YARD S/S April 87

[21] A. Tilbury, "KNOSOS Typical Configuration System data and Problems from YARD point of view", KNOSOS Document 29/Y/Rev0, YARD S/S, April 87

[22] J.-P. Hubaux, J.-P. Quillien, "A Typical Industrial Problem", KNOSOS Document 38/A/Rev1, ALCATEL, Juin 87

[23] B. Meijer, E. Rames, "Example, description and use of SIBEMOL/ROSACE", KNOSOS Document 33/K/Draft, MATRA Espace, May 87

[24] F. Billich, "DORNIER EXAMPLE for Test and Evaluation", KNOSOS Document 39/D/Draft, DORNIER, Juin 87

[25] J.-L. Gregis, R. Valent, "Evaluation of ROSACE, Example of an Advisor for ESI S/W Project", KNOSOS Document 34/E/Rev0, ESI, June 87

[26] B. Meijer, "Definition of a Component Model", KNOSOS Document 36/M/Rev0, MATRA Espace, June 87

[27] R. Valent, "Glossary of Header Terms", KNOSOS Document 40/E/Draft, June 87

[28] Guy L. Steel, "Common-lisp : The Language", Digital Press, 1984

[29] PCTE, ESPRIT Project number 32

Project No. 1262

SFINX: Towards PCTE based Software Factories

ABSTRACT:

The aims of SFINX are to:

* Prepare for the emergence of industrial PCTE based Software Factories through the realization of prototypes obtained by combining and integrating on top of PCTE the results of the ESPRIT projects in the Software Technology and Advanced Information Processing areas.

* Contribute to the promotion within the European community of the ESPRIT results in the above mentioned areas, through the realization of user oriented documents and training course supports.

The project started September 1st 1986. It is a five years project.

The definition phase, which just finished, allowed, in a one year time:

* to produce techniques and tools for qualification and integration of the various components of the Software Factory,

* to settle down the technical infrastructure for experimentation and evaluation,

* to initialize the promotion activities.

The second phase will correspond to the working up of the results of the definition phase to build, experiment and promote prototypes of PCTE based Software Factories.

Five collaborating European Companies are involved in this project:

CAP (UK), CRI (DK), CSATA (I), ERIA (SP) and SFGL (F).

This paper first explains why SFINX is existing. It then describes rapidly the SFINX strategy in order to introduce two of the key activities conducted in the project: qualification and integration.

After a more detailed explanation of what SFINX has realized within these two domains, it ends with a presentation of the SFINX longer term objectives and expected following up.

1. The SFINX Project

SFINX, Software Factory INtegration and eXperimentation, is an ESPRIT project in the Software Technology sub-programme. It aims at furthering the emergence of advanced Software Engineering Environments, identified as Software Factories, based on PCTE (a basis for common tool environments) by using tools developed within the frame of ESPRIT in the Software Technology and Advanced Information Processing sub-programmes.

SFINX is the logical following-up of the other on-going ESPRIT actions in Software Technology.

PCTE provides a set of powerful, integrated and homogeneous facilities to support the development, the integration and the use of software engineering tools.

The Software Technology and Advanced Information Processing sub-programmes will result in the availability of software engineering tools supporting different models of software development process and implementing different methods.

SFINX has three objectives:

* to verify, improve and promote the usability and availability of PCTE as a basis for Software Factory implementation,

* to demonstrate that the tools produced at least by the two sub-programmes can be combined in various instantiations of PCTE based Software Factories,

* to add to the dissemination and exploitation of the results of the two sub-programmes and contribute to their industrialization.

It provides a suitable framework for the qualification, integration and promotion of tools as well as for experimentation with instantiations of Software Factories. Within this framework the usefulness of PCTE in an industrial context and the emergence of Software Factory is effectively demonstrated.

As ultimate goal, the SFINX project will enable a Software Producing Enterprise/Company clearly to identify its requirements for a specific instantiation of a Software Factory. Such identified requirements, which are strongly dependent on the type of product to be produced, the type of market to be targeted and the quality of staff available, will facilitate the selection of the appropriate methods and tools as well as the identification of the required level of integration. Emphasis is placed on integrating tools at the highest practicable level.

The result will be highly valuable when targeting a common methodology on tools used for software development and when formalizing the evolutionary incorporation of a comprehensive set of standard interfaces and methods for integration.

SFINX was launched in the Autumn of 1986. It is a five years project. The effort estimated is approximately 900 men-month. The project partners are:

* CAP (UK),
* CRI (Denmark),
* CSATA (Italy),
* ERIA (Spain),
* SFGL (France).

SFGL is the prime contractor and has subcontracted to VERILOG for years 2 to 5.

2. The SFINX Strategy

2.1. The S.F. concept

The SFINX Strategy is based on the Software Factory concept. This concept must be considered as the necessary glue between the various activities constituting the different phases of the project. It is the vehicle for achieving their coherence and to reach a common understanding between the SFINX partners.

A preliminary generic model of Software Factory has been defined. This model will be updated throughout the whole life of the SFINX project according to the experience gained in building instances of Software Factories as well as to the findings based on theory. This model will also be improved by following the work of other non ESPRIT projects like the Alvey ISF project or the Eureka EAST and ESF projects. The SFINX Software Factory model accounts for both technical and organizational constraints and for the coherent organization of functionalities pertaining to different software methods.

Using such generic model as a framework, different instances of the Software Factory will be deduced. They will be characterized by different methods for software development and by different life cycle models, servicing different requirements as resulting from the software to be produced as well as the type of market the produced software will be targeting. Further, the quality of staff, their level of education, and the level of component reuse (i.e. the level of industrialization) will be taken into account.

The instances of the SFINX Software Factory will be realized on top of PCTE.

2.2. The Project Phases

The elaboration of the SFINX strategy has allowed the identification of three different phases.

* A definition phase, which corresponds to the first year of the project. During this phase, all the required methods, mechanisms and tools necessary to achieve the other activities constituting the project have been defined and

specified. In particular a first generic model of S.F.

* An implementation phase: selected tools from the Software Technology, Advanced Information Processing and eventually the Office Automation sub-programmes are qualified and integrated to realize specific instances of the SFINX Software Factory. Promotion of the results is ensured at all stages of this process through the delivery of documents, training materials and demonstrations.

* An experimentation phase: experimentation with several Software Factory prototypes, even if they correspond to incomplete instantiations of the S.F. model, will provide the basis for the creation of demonstration centers.

The first phase has been more dedicated to theoretical and conceptual work while the two other phases are more dedicated to experimentation.

2.3. A pragmatic approach

One of the main characteristics of the selected approach is to be pragmatic:

* On one hand, all the theoretical results will be refined and completed all along the project duration thanks mainly to the results of Experimentation: Experimentation of tools, results of integration, Experimentation of S.F. instantiations.

* On the other hand the S.F. prototypes will not be defined starting from specific user needs or requirements and then implemented, after having selected or developed the appropriate tools.
  The S.F. prototypes will be defined and implemented depending on the characteristics of the available tools coming from other ESPRIT projects and therefore depending on the results of the tool Qualification.

2.4. A Historic Parallel to the SFINX Scenario

The Software Factory Experimentation based on ESPRIT tools may be compared with the early stage of development of techniques for wide area communication (the pre-OSI stage). As a result of such experiments, it was possible in 1977 to set up activities resulting in a reference model for Open System Interconnection. During the first two years, the model was changed from five to seven layers in order to accommodate some more structuring at the application level and in 1979 the ideas were stabilized, even though a stable description was not developed before 1982. The reference model was standardized in 1984 and specific standards for the different layers are now in the process of being developed. In 1982 some new concept of a functional profile was identified and work is now in progress on an international level to implement such ideas.

The present stage of Software Factories and Software Engineering

Environments is to be compared with the pre-OSI stage.

The work of the SFINX project is aiming at on a pragmatic basis and on the basis of the existing ESPRIT tools to cater for the transmission into formalized work on the Software Factory concept. This will be carried out by targeting a common integration-oriented taxonomy on tools used for software development and by aiming at formalizing the evolutionary incorporation of a comprehensive set of standard interfaces and methods for integration.

2.5. SFINX Activities

The key activities of the project are Qualification, Integration, Experimentation and Promotion.

Qualification will lead through a better knowledge of the considered tools to the definition of different prototypes of specific instances of S.F. which will be implemented and experimented. It leads also to the integration process in the sense that it identifies the tool's behaviour at the sites within a given instance of S.F. where integration can take place.

Integration will result in the physical implementation of prototypes of may be incomplete instances of S.F.. Qualification and integration activities are supported by rules, mechanisms and tools elaborated during the definition phase according to the SFINX Software Factory model.

Promotion, in the SFINX strategy, supports the dissemination and exploitation of the ESPRIT results as they are coherently organized into prototypes of Software Factory . Promotion actions include the production of user oriented documents and manuals, and computer aided and video training supports.

Experimentation within the project with several Software Factory prototypes will provide the basis for the creation of demonstration centres.

The SFINX project builds prototypes of Software Factories, and thus provides a work bench for Experimentation with the development of Software Factories. A SFINX Software Factory has a PCTE kernel and consists of tools resulting from the ESPRIT programme.

The SFINX Experimentation should result not only in extended knowledge on PCTE and the tools in question, but also on Software Factories and to some understanding on common requirements to Software Factories.

Achievement of the SFINX results largely depends on the collaboration with other ESPRIT projects: specific agreements will be established in the frame of a general collaboration scheme.

Results from SFINX qualification, integration and promotion activities, together with assistance in porting tools on top of PCTE, will constitute a valuable return value for the collaborating projects and a decisive step toward the

industrialization of their results.

In short, the SFINX project provides the software production community with a PCTE based Software Factory work bench for integration and experimentation.

3. Qualification

3.1. Objectives

Qualification in the SFINX strategy has two kinds of objectives:

* characterization of a tool in respect of its users, describing what functionalities it covers and how it provides them; this kind of qualification is necessary in order to locate the tool in its pertinent instance of Software Factory;

* evaluation of a tool according to integration metrics, in respect of its PCTE based operating environment specifying and quantifying the way the tool uses it, in respect of the other tools and in respect of its users.

Qualification leads, in SFINX, to the integration process, in the sense that it identifies the tool's behaviour at the sites where integration can take place.

Qualification is supported by a "Tool questionnaire" and Qualification tools.

3.2. Mechanisms and tools

The objective of the "Tool questionnaire" is to provide standard characterization of the tools and the approach is to determine how and where a tool fits into a Software Factory. Thus, the tool will be characterized according to the concepts used in this paper, namely as life cycle, methodology and application area.

Mere combination of tools does not lead to a Software Factory. If the activity of combining tools shall lead towards a Software Factory, two aspects of integration are to be considered, technical integration and uniformity.
Technical integration is investigated through characterization of the tool interfacing to its surrounding environment.
Uniformity is investigated through the determination of the user interface characteristics of the tool.

Qualification tools include a PARSER and a TRACER, they provide help (support) for the analysis and the evaluation of the way the tool is using the operating environment.

The PARSER is doing a static analysis of the source code of a tool and produces a list showing the used devices of the operating environment.

The TRACER

* captures all or a user specified portion of the information flowing between the tool and the operating environment (PCTE)

* provides statistic information on the actual usage of PCTE primitives.

These informations are gathered during tool execution and subsequently edited.

3.3. Activities and Results

Two levels of Qualification are considered:

* Qualification based on paper study

* Qualification based on Experimentation (real use of a tool).

Each of the SFINX partners has in charge the Qualification of a certain number of projects. Paper studies driven by the "Tool questionnaire" are performed in parallel on the different partners' sites.

Results of this first level of Qualification are then gathered and analysed in order to determine, starting from what can be considered as a disordered set of tools, functional ordered subsets, eventually partly overlapping, each of them corresponding to an incomplete instantiation of Software Factory. This can also be considered as a way to determine the feasibility of the integration of the tools within a Software Factory from a conceptual or functional point of view.

A second step: Qualification based on Experimentation still performed in parallel on the different partners' sites allow to refine the answers to the "Tool questionnaire". It also implies the use of the Qualification tools. Results of such Qualification should allow to determine the feasibility of a tool integration from a technical and external (User Interface) point of view. It should also provide information about the level of integration which can be expected and help to select appropriate integration mechanisms.
The results of this second level of Qualification will then be gathered, analysed all together with the results of the first level, they should allow the identification of the prototypes of Software Factory instantiation which could be (at least partly) realised within SFINX, starting from the available tools coming from collaborating projects.

4. Integration

4.1. Objectives

Integration within the SFINX strategy corresponds to the real implementations of the prototypes of Software Factory identified thanks to the results of the Qualification activity.

Integration includes two types of activities:

* porting to PCTE,
* integration on top of PCTE.

4.2. Techniques and mechanisms

These activities may imply the use of one or more different mechanisms depending on the status of the considered tool, depending on its role and location in the prototype, depending on the level of integration which is foreseen.

The level of integration depends on the type of relation which is established between the tool and:

* the operating environment,
* the other tools outside the software life cycle or in the software life cycle,
* the external world: systems outside the Software Factory, the Human Computer Interface (HCI).

The integration techniques and mechanisms consist of:

* rules to be followed during the development of the tools
* software support
  - software support for using the rules (i.e. use of common services layer interface),
  - Basic software support to be used within the project to demonstrate that tools coming from different projects are able to cooperate even if they have not been designed taking into account the above mentioned rules.

The identified techniques and mechanisms are the following:

A) tool for making known a tool, its functionalities, its interfaces

B) tool which supplies a layer making a tool independent of the appearence of the HCI to the user

C) PACT gateway toolkit

D) a systems analysis of the software development process

E) PACT metaschema

F) Remote procedure call abstraction

G) Interface Data Language and data conversion by stages

H) PACT tool composition

5. Final results of the project and expected following-up

Qualification of tools in regard of their relationship with the operating environment, Evaluation of their integrability within PCTE based Software Factories will provide the elementary inputs for the elaboration of a guide for PCTE based tool developers.

The "Tool questionnaire" improved thanks to the results of its use or more generally improved through the resluts of the Qualification, Integration and Experimentation should serve as a basis for the elaboration of guidelines for tool selection i.e. guidelines for Software Factory builders helping them to select and integrate the methods and tools the more appropriate to their needs and requirements.

Therefore SFINX should contribute actively to the emergence of PCTE based Software Factories. Such an action will be reinforced by the creation of a structure intending to serve as a forum for PCTE tool developers and more generally for PCTE based product developers.

Project No. 125

THE USE OF THE OBJECT-ORIENTED APPROACH IN THE GRASPIN DB

S.Goutas, P.Soupos, C.Zaroliagis, D.Christodoulakis, D.Maritsas

Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece

Electronic Mail: dxri@dias.uucp

1. Introduction

One of the main trends that appeared after the software crisis in the early 70s, was the development of software engineering environments (SEE). Pioneers in the field were the GANDALF and MENTOR projects [5,9]. The promising results of both projects led the CEC to include this area of research in the ESPRIT programme. One of the first such projects in ESPRIT, was GRASPIN which aims at the construction of a personal software development workstation dedicated to support formal specification and stepwise implementation of large software systems.

The various tools of a SEE need a common representation of data in order to cooperate and thus provide an integrated environment. This is achieved by using a Software Engineering Database (SEDB). SEDBs must in general fulfill the following requirements:

- there must be an appropriate database model capable of representing programs and their semantics at the same time
- there must be a uniform and flexible query language in order to access the various types of information stored, and finally
- the need for fast transactions with the database since we deal with interactive environments.

The most important of the above is the data model which determines the abstract view of data and defines a framework of concepts that can be used to represent programs, program semantics and program development histories. For the best expression of the above, the database model must provide a one-to-one mapping between whatever is thought as an entity in a SEE and a database entity. Since a user's application in a SEE may consist of several entities with a rather complex structure, conventional database models (relational, network, hierarchical) are not appropriate because they are designed to model mainly commercial data with lower complexity structure [1,7,11,13].

Existing SEDBs [5,6,8,9] handle two types of information: the program itself and the semantic information about it. The standard approach for program storage and retrieval has been their representation as abstract syntax trees (ASTs), while semantic information is handled in a variety of ways. For example, in [5] program semantics is handled using action routines (demons), and in [6] a hybrid storage system is proposed where programs are stored as attributed ASTs and program semantics is organized using relations. Although the former is quite powerful it does not actually organize and store the semantic information in the database. The latter solves the above problem, but an intergrated database model is not provided to avoid incompatibilities among the various data structures used.

The aim of this paper is to present the GRASPIN DB, which is the SEDB of the GRASPIN environment. For the development of the database the above requirements and observations have been considered. The GRASPIN DB is object-oriented [18]. It provides a powerful object-oriented interface both to the users and the tools of the GRASPIN environment, while for the internal representation of programs and program semantics it utilizes widely accepted concepts such as attributed ASTs and relations [5,6,8,9]. The main advantage of the object-oriented interface is that it provides not only a unified view on both programs and program semantics, but also supports information hiding, data abstraction and inheritance. Furthermore, the object-oriented approach meets most of the requirements discussed in [13] and provides a one-to-one mapping between a database entity and a SEE entity.

The object-oriented interface is integrated with an object-oriented query language providing methods for the retrieval and manipulation of objects. The query processor developed for that purpose, serves the GRASPIN SEE as the only communication/transaction mechanism between the environment and the database. Therefore, the handling of database objects is strongly uniform and independent of their internal structure. Basic feature of the query language is the ability to express queries that operate on both ASTs and relations, thus providing a homogeneous mechanism for transfering information from ASTs to relations.

This paper is organised as follows: section 2 presents an overview of the conseptual architecture, section 3 the physical organisation, and section 4 the query language. All figures are included in the Appendix.

2. The Data Model of the GRASPIN DB

From a conceptual point of view, the GRASPIN DB is viewed as a collection of entities that correspond to the several objects generated by the GRASPIN environment. These entities are organized into classes that correspond to the entity-types whose instances are the particular entities.

The *class* construct is the basic building construct of the GRASPIN DB. A class specifies a particular data structure for the storage of any entity-type existing in the GRASPIN environment (document, program, subprogram, diagram, etc.) and a set of methods that manipulate this entity-type. In this sense a class is the data structure which implements a set of class instances (objects) with a set of methods for handling the objects of that particular class. The user cannot see the internal representation of a class, he can only manipulate it by calling the associated methods. Objects of a particular class which have further similar properties can be grouped in certain collections named *subclasses.*

The other basic building construct in the GRASPIN DB is what is called in [12] as *system classes.* These classes play an important role in the GRASPIN DB since they are its basic building blocks. The whole database inherits the basic types and methods from the *system classes* that constitute the basic kernel system. The most important system classes are the so called TREE and RELATION. They are used for the internal representation of programs and their semantic information, respectively. These system classes contain the appropriate data structures and methods for the storage and retrieval of ASTs and relations. Another system class is the class OBJECT which plays the important role to be the superclass of all classes generated in the GRASPIN DB. Thus, the data structures and methods provided either by itself or by the other system classes are inherited to the other database classes.

Figure 1 of the Appendix shows an overview of the conceptual architecture of the GRASPIN DB. As you can see, GRASPIN is a multilingual environment supporting a certain number of Language Environments (LE). A LE is an interactive syntax-directed SEE dedicated to a specific language; e.g. the syntax-directed SEE for Pascal programs constitutes the Pascal LE. Each LE forms a database class, called LE-class. Such a class is created by the *Transformer* which is a tool of the GRASPIN environment that takes as input the specification of a LE in terms of the language description language ASDL and creates the corresponding data structures into the database [4]. The main role of the ASDL is to serve as the formalism which defines a complete translation scheme for a LE [10], and therefore constitutes the Data Definition Language of the GRASPIN DB. In other words, the GRASPIN DB schema is completely specified by the ASDL description. The main parts of an ASDL description are: a) the description of the grammar of the particular LE which is used for the storage of the ASTs, b)

the definition of relations for handling semantics and c) the definition of the semantic actions, which operate on ASTs and relations and serve among others the information transfer from ASTs to relations and vice-versa. Like every other language, ASDL has some predefined types (e.g. number, string, list) and some predefined actions. As denoted in [10], both of them can be imported from somewhere outside the ASDL. In our case, the predefined types and actions in ASDL specifications of LEs are provided by the database system classes.

To clarify the above, let us consider the ASDL module of figure 2 which describes a simple fragment of a Pascal-like LE. Grammar rules, declared in the *types* part, give the conceptual definition of the ASTs. The *attributes* part defines the attributes attached to the symbols of the grammar and are used for the representation of semantic information about programs as introduced in [14]. The *relations* part gives the definition of two relations which are used for dataflow analysis purposes. Relation DEF interrelates program variables and the corresponding program point where they are defined. In a similar way, relation USE interrelates variables and the program point where they are used. Finally, the *actions* part gives the definition of semantic actions operating on ASTs and relations. Actions are described in a formalism equivalent to that of the CDL2 language [15]. The action "repl_and_eval" is based on some predefined actions that will be explained in the next section. This action is invoked during editing of a program to update the attributed AST and the corresponding relations. For more details about ASDL the intrested user is refered to [10].

The transformer creates from the ASDL specification, the corresponding LE-class with the following data structures: 1) a directed graph representing the grammar of the LE and 2) a set of relational structures each one corresponding to a particular relation which has been defined in the LE specification. Furthermore, the transformer generates the LE-classs methods from the corresponding semantic actions. The data structures and methods of the LE-classes are used by the tools of the environment to create or manipulate the ASTs and the relations. Programs are stored as instances of the corresponding LE-classes.

We conclude this section with some basic well-behaveness conditions required to generate well-formed schemata from ASDL descriptions.

1. Each ASDL specification of a LE must define exactly one LE-class in the GRASPIN DB.
2. Each predefined type of the ASDL must correspond to one and only one data type defined in the system classes of the GRASPIN DB.
3. Each predefined action of the ASDL must correspond to one and only one primitive method defined in the system classes of the GRASPIN DB.

3. The Internal Level of the GRASPIN DB

In this section the internal level of the GRASPIN DB is presented. This part of the database is crusial regarding efficiency, portability, consistency, e.t.c.

The internal level consists of three layers. The first is the database file management system, the second layer deals with the data structures namely hierarchies and relations and the third is the object interface.

3.1 The file management system

For reasons of portability, restrictions on multi-user filling on some host mashines, consistency e.t.c., the GRASPIN DB is stored in one single file of the host, with random access records of fixed size [16]. These records are called *pages*. The DB file system deals with the internal management of this file. It is a hierarchical file system consisting of file-directories and files. Directories and files alike are stored on disk in a set of pages. The root of the whole file/directory structure within the DB file is the *Master directory*, used for bootstrapping the DB, which contains an unordered set of pointers to every directory.

3.2 Hierarchies

The abstract syntax of a LE determines the directory hierarchy as well as the internal structure of the files that facilitate the storage of the ASTes. This hierarchy is implemented by establishing pointers between directories. Thus the internal structure of the DB is syntax directed. Every AST is a collection of two tables; a *syntax table* that contains the context free structure of a document as a tree of nodes that contain context dependent attributes and a *symbol table* that contains tokens and other lexemes required for AST processing. Both tables are stored in separate files.

Every directory contains abstract syntax trees or subtrees of the same type. For example there is a directory for the authors of a LE, a directory for the programs of that author e.t.c., as you can see in figure 3 where the following grammar subset is used:

```
* environment ::= author[author]
  * author ::= program[program]
    * program ::= module[module]
      module ::= stmtlist[stmtlist]
```

The asterisk denotes the creation of a directory for the subtree that corresponds to the respective nonterminal. The root of the directory hierarchy of a LE is the *Root directory*. The abstract syntax of a LE is also stored in the DB in a separate directory as an asyclic graph. This graph is used by various syntax directed tools of the GRASPIN environment.

3.3 Relations

As it has been mentioned, relations are aggregations of attributes over ASTes. In that sense the relational structure is interwoven with the hierarchical one. Relations too consist of two tables; a *relation table*, that holds tuples as sets of entries of equal size and a *symbol table*, that contains the values of tuple fields.

As it has been mentioned in the previous section, relations are declared in the ASDL description of the grammar of a LE and furthermore relations are associated to grammar nonterminals. Thus relations are stored according to the nonterminals they are associated with in the corresponding directory. Each table again occupies a separate file.

Since the number of tuples of a relation is unknown and changes with time, the size of the relation table changes dynamically according to the storage needs. For that purpose a variation of linear hashing [17] has been used.

3.4 The Object Interface.

As mentioned in the introduction the GRASPIN database model is an object oriented one. The database objects are either trees or relations. The tree objects' granularity is defined in the grammar description that parameterises the database. Relation objects on the other hand are single reltions. At the physical level the AST and the relations that belong to a certain directory relations are objects. Therefore the DB objects are not a meta structure of a high abstraction level of the database but they exist at the physical level as well. This is an important point as far as efficiency is concerned.

The *object interface* is a set of primitives at the physical level for easier and more efficient manipulation and maintainance of objects in main memory. The object interface is the interface of the database low level structures to the object related database tools. It basically constructs two data structures, that of the object and that of the *object table*. The object is a simple structure that contains a pointer to its relation or tree, it is created whenever a tree or relation is loaded in memory. All objecs are stored in a memory area called the heap. An object can be accessed by a pointer from other objects, when there are no references to an object this object is deallocated. The object table associates pointers to objects with their real address in memory or disk.

4. The External Level of the GRASPIN DB

The object oriented interface of the database is intergrated with an object-oriented query language which, through its predefined methods, enables the retrieval and manipulation of objects regardless their internal structure. Thus, the query language provides a homogeneous mechanism for transfering information from ASTs to relations. Furthermore the query language provides uniform access to the conceptual entities of the environment, (eg. programs, users, dates etc.) since it is the only communication/transaction mechanism between the database and the rest of the environment.

The query language has a message passing mechanism similar to the one introduced in Smalltalk-80 [12] and a formalism similar to that of ASDL, in order to maintain uniformity with the DDL of the database. Queries are defined and stored as methods in a special class of the environment since they constitute the actual interface of the database to the environment. The methods defined in the system classes constitute the kernel of the query language. That is, they are the primitive queries that enable the retrieval and the manipulation of the objects of some predefined data type. More complicated queries may be defined invoking either the primitive queries or others that have been defined previously.

The protocol of the system class TREE consists of methods that make possible the creation and manipulation of an AST as well as the evaluation of its attributes. More precisely, these methods can access a node or a part of it in an AST, traverse an AST, evaluate its attributes, or replace subtrees of an AST with new subtrees, etc. The most important of those methods are presented in figure 4. A derived or transient parameter in a method denotes the receiver of a message, while inherited parameters denote the arguments. To understand their use, consider the following example.

EXAMPLE 1: Assume that after editing a program, its AST is stored in the database as an object T. Suppose now, that the user wants to replace a part of his program text that corresponds to the subtree S1 of T. The replacement of S1 with the new subtree S2 corresponding to the new text added, causes the incremental evaluation of attributes which have been changed due to the replacement. This is done using the following query which can be defined and stored as the body of a method in the database class.

```
(1)    replace(T,S1,S2)
(2)    propagate(T,S2)
```

Line (1) of the above query means that object T receives the message "replace" by invoking the corresponding method from the class TREE. This results to the replace-

ment of the subtree S1 with the subtree S2. The replacement of the subtree S1 leads to changes in the attribute values of T and thus reevaluation is needed. The propagation of changes in attribute values is done in line (2) where the object T (after the tree replacement) receives the message "propagate". This method corresponds to the well known approach for incremental attribute evaluation, called change propagation. □

The protocol of the system class RELATION enables the manipulation of relations. That is, it provides the basic relational operators (union, minus, cross, select, project), allows the definition of a new relation as well as the updating of a new relation with the value of a computed one. Figure 5 includes some methods of the class RELATION. The following examples help to clarify their use.

EXAMPLE 2: Consider the relation DEF(Pp,Var), shown in figure 2, which associates each variable of a program with the program point where it is defined. A possible query of interest could be: "Find the program points where the variable A is defined in my program". To answer the above query, we write:

```
select(DEF,(Var='A'),DEF1)
  project(DEF1,Pp,DEF2)
```

The relation DEF1 receives the message "select" which selects those tuples from DEF that meet the condition Var='A'. Consequently, the relation DEF2 is the receiver of the message "project" which projects the relation DEF1 on Pp. □

Apart from the above queries, it is possible to express queries that operate on both ASTs and relations. A useful kind of such queries is the one that aggregates the semantic information from an AST and stores it to a certain relation. This is presented in the following example.

EXAMPLE 3: Assume again the relation DEF(Pp,Var). The following query traverses an AST and updates accordingly the relation DEF:

```
update(object>DEF>,object>T):
                    getroot(T,Root),
                    ( visitnextnode(Next,Root), get(Next,P),
                                        get(Next,V), maketuple(Tuple,P,V),
                                        entertuple(DEF,Tuple), make(Root,Next),
```

□

5. Epilogue

The first prototype of the GRASPIN DB has been implemented on VAX 750/VMS in CDL2 and LISP and has been ported to the Symbolics 36xx systems. The final prototype of the GRASPIN DB as well as its intergration with the other tools of the GRASPIN environment will be achieved during the enhancement phase which has already started.

References

[1] Dadam P., et al., "A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies", Proceedings of SIGMOD 86, page 356.

[2] Zaroliagis C., Soupos P., Goutas S., Christodoulakis D., "The GRASPIN DB - A Syntax Directed, Language Independent Software Engineering Database", Proceedings of the 1986 International Workshop on Object-Oriented Systems, page 235, Pacific Grove, CA, Sept 86.

[3] Soupos P., Goutas S., Christodoulakis D., Zaroliagis C., "The GRASPIN DB - A Software Development Environment Database", ACM SEN, Vol.12, N.1, Jan 87.

[4] GRASPIN Project Team, "Architecture of the Final GRASPIN Workstation Prototype", Technical Paper GRA 80/2, October 1986.

[5] Notkin D. et al., Special Issue on the GANDALF Project, In the Journal of Systems and Software, vol.5, No.2, North-Holland, 1985.

[6] Horwitz S., Teitelbaum T., "Relations and Attributes: A Symbiotic Basis for Editing Environments", Proc. of the SIGPLAN 85 Symp. on lang. issues in prog. environments, Seattle, WA, pp. 93-106, June 85.

[7] Linton M.A., "Implementing relational views of programs", Proc. of the ACM SIGSOFT/SIGPLAN software eng. symp. on practical sofware development environments, Pittsburgh, Penn., April 1984.

[8] Stahl M., et al., "Documentation for the CDL2 Lab", November 1985, Report No 72, Katholieke Universiteit Nijmegen.

[9] Donzeau-Gouge V., et al., "Programming Environment Based on Structure Editors: The Mentor Experience", June 1980, Workshop of Programming Environments, Ridgefield, Ct.

[10] Christ-Neumann M.L., et al., "ASDL-A Specification Language for Syntax-Directed Environments", GRASPIN Technical paper GMD16/5, February 1986.

[11] Dittrich K. R., "Object-Oriented Database Systems: The Notion and the Issues", Proceedings of the 1986 International Workshop on Object-Oriented Systems, page 2, Pacific Grove, CA, Sept 86.

[12] Goldberg A., Robson D., "Smalltalk-80: The Language and its Implementation", Addison-Wesley, 1983.

[13] Imperial Software Technology, "Requirements for Software Engineering Databases", Final Report, June 1983.

[14] Knuth D., "Semantics of Context-Free Languages", Mathematical Systems Theory, Vol.2, No.2, Springer-Verlag, New York, 1968.

[15] Koster C.H.A., "Draft on a Textbook on CDL2", Informatics Dept., Nijmegen University, The Netherlands, 1983.

[16] Dehottay J.P., "Preliminary description of SEEK's Syntax Oriented Database", Nov. 1985.

[17] Litwin W., "Linear hashing: A new tool for file and table addresing", I.N.R.I.A.

[18] Christodoulakis D., Soupos P., Zaroliagis C., "The Implementation of a Software Engineering Database Using Desk-size Computing Resources", to appear in the Proceedings of EUROMICRO 87, 13th Symposium on Microprocessing and Microprogramming, Portsmouth, England, Sept 87.

Appendix A
Figures

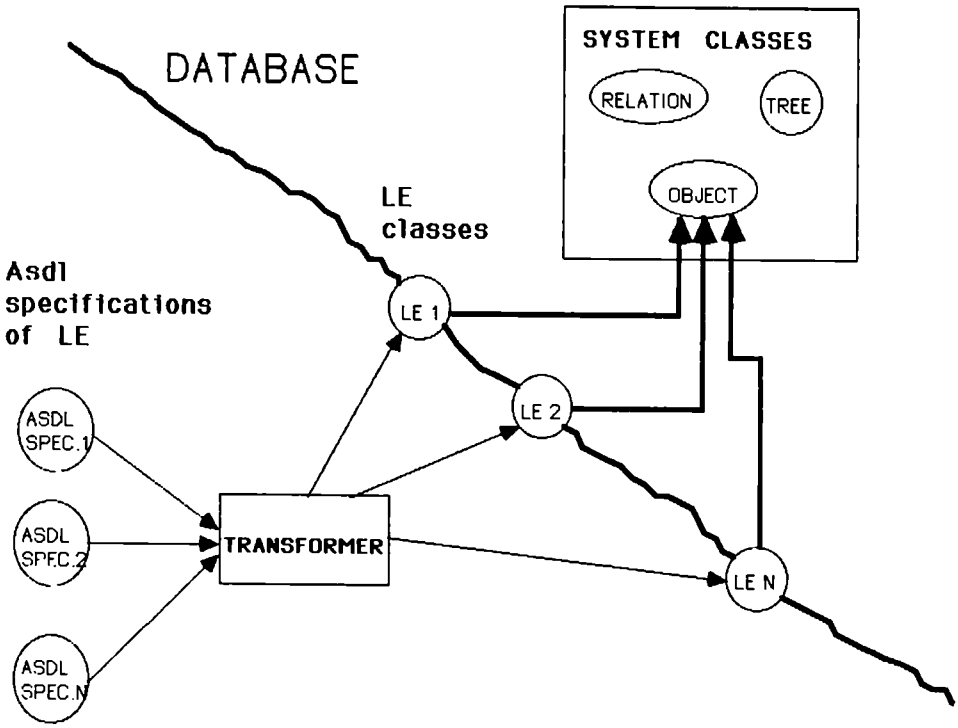


Fig.1

```
Module Pascal-like LE fragment.

    types.

    assignment   : id,expr.
    expr         : number;
                   id;
                   expr;
                   arlogexpr.
    arlogexpr    : expr,op,expr;
                   unop,expr.
    op           : .
    unop         : .
    id           : .
    number       : .

    attributes.

    name : id ⟶ $string.
    pp1 : id ⟶ $num.
    pp2 : assignment ⟶ $num.

    relations.

    DEF($num>Pp,$string>Var).
    USE($num>Pp,$string>Var).

    actions.

    repl_and_eval(object>T>,object>T1,object>T2) :
                  $replace(T,T1,T2), $propagate(T,T2),
                  $update(DEF,T), $update(USE,T).

endmodule.
```

Fig. 2

The attribute name is on the left of the semicolon, while the associated grammar symbol is on the right. The attribute type is denoted on the right of the symbol ⟶ , while the dollar sign ($) means that the corresponding attribute type or action is imported. The notation >x means that x is an inherited (input) parameter, x> means that x is a derived (output) parameter and >x> means that x is a transient (inhereted and derived) parameter.

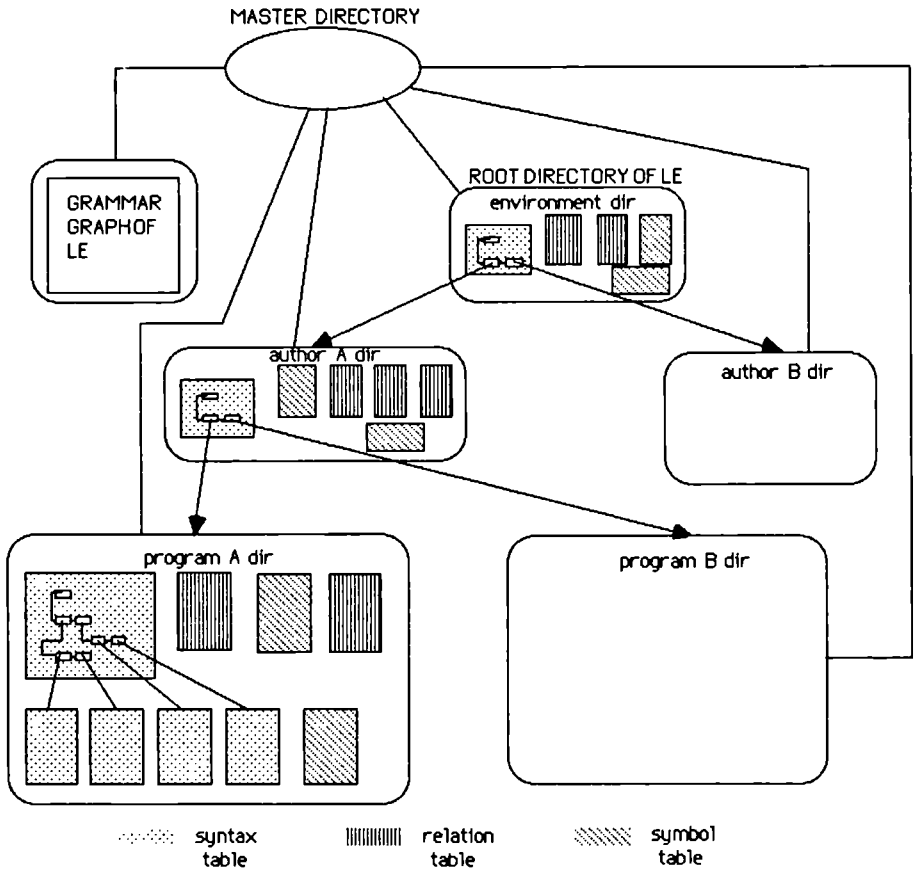


Fig. 3

Methods	Semantics
getroot(object>T,R>object)	finds the root of an AST T and returns its pointer to R
visitnextnode(N>object,object>R)	visits the next node of a node R and returns its pointer to N
replace(object>T>,object>T1, object>T2)	replaces the subtree of the receiver labeled T1 with the subtree labeled T2
propagate(object>T>,object>S)	makes the propagation of attribute values at node S of an AST specified by the receiver
evaluate(object>T>)	evaluates the attributes of an AST specified by the receiver

Fig.4 Protocol of the system class TREE

Methods	Semantics
maketuple(T>object,object>a1, ...,object>an)	creates a tuple T (receiver) consisiting of values specified by the objects a1,....,an
entertuple(object>R>, object>T)	enters the tuple T to the receiver specified by the relation R
union(object>R1,object>R2, R>object)	creates a new relation R which is the receiver of the message union of the relations R1,R2
minus(object>R1,object>R2, R>object)	creates a new relation R which is the receiver of the message minus of the relations R1,R2
cross(object>R1,object>R2, R>object)	creates a new relation R which is the receiver of the message cross product of R1,R2
select(object>R1,(cond), R>object)	creates a new relation R which is the receiver of the message select that select those tuples from R1 which meet the condition
project(object>R1,object>A1, ...,object>An, R>object)	creates a new relation R which is the receiver of the message project that projects relation R on attributes A1,...,An

Fig.5 Protocol of the system class RELATION

Methods	Semantics
get(object>N,>P>object)	returns the value of a part of a node N specified by the receiver P, to the receiver
make(X>object,object>Y)	makes the value of object Y to become the value of object X

Fig.6 Protocol of the system class OBJECT

Project No. 401

ASPIS: A KNOWLEDGE-BASED ENVIRONMENT FOR SOFTWARE DEVELOPMENT.

F. Pietri, P. P. Puncello, P. Torrigiani
(Tecsiel S.p.A., V. S. Maria, 19, 56100 PISA - ITALY)

G. Casale, M. Degli Innocenti
(Olivetti S.p.A., D.O.R., Via Palestro, 30, 56100 PISA - ITALY)

G. Ferrari, G. Pacini, F. Turini
(Dipartimento di Informatica, Corso Italia, 40, 56100 PISA - ITALY)

This paper gives an overview of the ESPRIT project 401, which aims at building a knowledge-based software development environment. Its main novelty is the exploitation of AI techniques in order to build tools called Assistants and to define an executable logic-based specification language. The current status of the project is described

0. Introduction

The main goal of the ASPIS project is to exploit AI techniques in order to build a software development environment supporting a more flexible and effective life cycle characterized by smooth transitions between user needs, specification, and design.

The ASPIS environment encompasses a set of tools specifically addressing tasks which are in the earliest phases of software life cycle. Indeed, those phases (requirements analysis and design) are very knowledge-intensive and require the expertise of skilled practitioners. The processes which convert requirements into a specification and then into a detailed design, are often informal, labor intensive and largely undocumented. It is elusive with the state-of-the-art techniques to plan to fully automate them. However, Artificial Intelligence techniques can contribute to the improvement of the quality and the productivity in those phases in two ways:

• they can provide techniques for building tools which augment the productivity of traditional software construction [1].

• they can provide new and higher level programming paradigms (e.g. declarative, i.e. rule based, programming), thus simplifying the whole job of building programs [2].

An evolutionary life cycle model has been adopted in ASPIS with the purpose of bridging the gap between the Analysis and Design phases. Suitable languages and methods have been established for both Analysis and Design and some application areas have been investigated with respect to these two stages (Access Control Systems, Business System).

The most novel aspects of the project are the development of knowledge-based tools called Assistants and the definition of a logic-based formalism, called Reasoning Support Logic, RSLogic for short, for the specifications. Such specifications may be executed by a Prototyper Assistant in order to verify immediately the properties of the system at hand and another assistant, the Reuse Assistant, will be built to help the developer in reusing old specifications and designs. These two assistants are called Support Assistants. The Knowledge-based Assistants are used directly by the developer when accomplishing the Analysis or the Design of a particular application in a given methodology. So they embody both knowledge about the method (Methodical Knowledge) and the knowledge about the application area (Domain Knowledge). The Figure 1 shows the four Assistants and their logical interconnections.

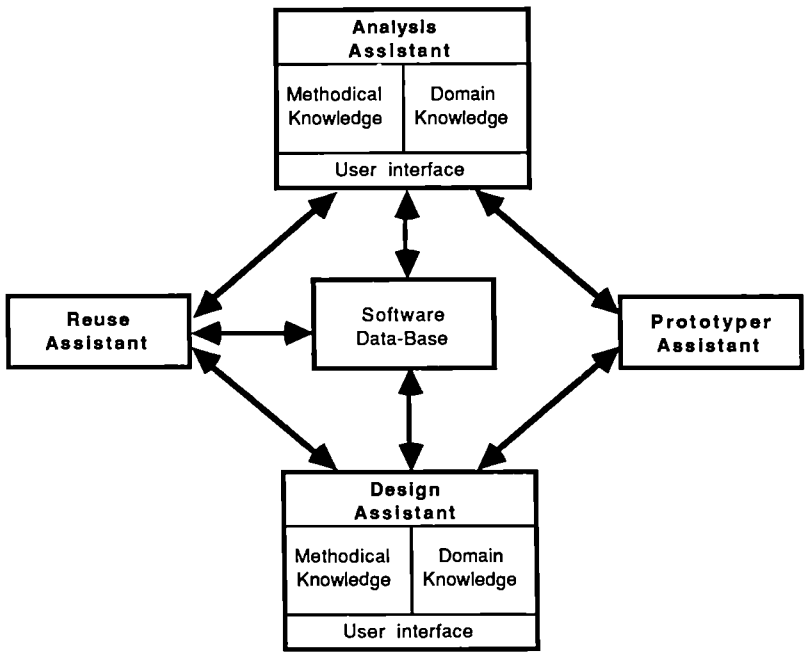


Figure 1

The paper will give an overview of the project and will mainly focus on the functionalities of the Analysis Assistant [3] along with examples and the techniques used for its development and on the formal description of RSLogic and its animation environment.[4].

1. History and Goals of the Projects

The project is split into two phases: a one year research phase (April 85 - March 86) and three years for development (April 86 - March 89). To date we have put our efforts in the definition of the ASPIS Software Life Cycle (SLC) and the formal specification language, in the investigation of Assistants' features and capabilities and in the design of the Knowledge-based Assistants. Currently

we are working on the development of a prototype which includes the Analysis and the Design Assistants along with very simple facilities for transforming the results of the former into the input of the latter. A knowledge representation system for this prototype has been built on top of Prolog exploiting meta-interpretation. This tool provide the possibility of defining semantic networks where the nodes can have also production systems as attributes [5]. Also a Prototyper Assistant, which allow to execute (or animate) the formal specifications, is under development.

For the final system a multi-layered expert system shell, which provides frame-like facilities and is implemented in Lisp, is under customization for the Aspis purposes [6] in order to offer a suitable interface with Prolog for prototyping. New versions of the Knowledge-based Assistants will be built along with the Reuse Assistant and a graphical User Interface. All the various tools will be suitably integrated in a development environment, which covers the SLC from the customer's needs to a detailed design.

2. An evolutionary life cycle model

The traditional approach to software development includes the definition of different phases or steps in a life cycle model, the definition of methodologies for each phase and the development of supporting tools, sometimes integrated in a sophisticated environment.

During the development process, different representations of a software system are produced in the various stages from particular viewpoints and with different purposes. Each representation must be complete and consistent with the others. For this purpose, a lot of research has been accomplished in the last years in the field of the formal specification languages [7,8,9].

The several existing life cycle models differ over the granularity of the stages, but most of them include the traditional phases of requirements analysis, design, implementation, test and maintenance. In all the SLC models the major bottleneck in the development process is the gap between specification and design. Moreover, Analysis and Design are now universally recognized to be the most crucial phases in the SLC, as the errors made in those phases have very heavy consequences on the successive steps of the software development. We have concentrated our focus on those phases and have tried to make such gap a smooth transition. We have picked and partially modified two different methods for the Analysis and the Design respectively and we have defined a way to move from one to the other stage.

The Analysis method is based on the DAFNE[1] methodology [10], which comprises several stages and intermediate results and uses the Structured Analysis (SA) language [11,12,13] for the analysis of functions and Entity-Relationship schema [14] for the analysis of data. The SA language is a structured but informal language which allows to describe a system in terms of functions (boxes) and data (arrows connecting boxes). In order to have the possibility of testing the correctness and the consistency of the specifications, a formalism to augment SA diagrams with semantic annotations has been defined. This formalism is called Reasoning Support Logic (RSLogic) [15,16] and allows to specify properties of a system via a set of axioms in a logic based language. Thus the ASPIS specification language is a specification language in which the SA graphic features are used for

1 DAFNE is a trademark of Italsiel S.p.A. and C.N.R.

representing the structural aspects while RSLogic takes care of the semantic issues. RSLogic can be considered both as a step towards a different way of building software, if one thinks that the execution environment can become so powerful that the specifications *become* the program, and as a way of enhancing the classical software life cycle with the possibility of performing rapid prototyping.
Indeed, rapid prototyping has been proposed as a powerful way of improving software productivity. The possibility of building a quick prototype, starting from the user requirements, allows to avoid the need of backtracking to the first phases of the life cycle, when the final software is tested.Our idea is that the prototype, i.e. the executable specifications of a software system, is defined to be a knowledge base, i.e. a collection of facts and rules describing the system. This approach presents two advantages:

1. The prototype is defined in a completely declarative way

2. The notion of executability of the prototype is generalized, in that a knowledge base can be queried in many different ways.

The Design has been split in design in the large and design in the small phases. The work to date has been concentrated on the design in the large. A language to describe a system in terms of processes and events [17] has been picked and conveniently modified in order to deal with features of real-time systems and the top-down approach has been adopted for the moment [18]. Other design methods might be adopted depending on the class of applications to be dealt.

Usually the designer needs to properly retrieve the specifications in order to simply read them or also to ask the analyst to modify them whenever he/she discovers they are not consistent. Furthermore, the customer's needs (and consequently the specifications) may change, and the design must be modified accordingly. The ASPIS SLC allows interaction between those two stages and comprises a process, called Synthesis, which can be seen as the gathering of information from the Analysis making it available in a suitable form to the Design phase and allowing an effective two-way communication between the analyst and the designer [19]. It is a continuous process as the designer needs to have the right information in several steps and to give feedback to the analyst. The Synthesis process includes also consistency checks between Analysis and Design on the basis of heuristics related to the application at hand. The Aspis development cycle is shown in figure 2.

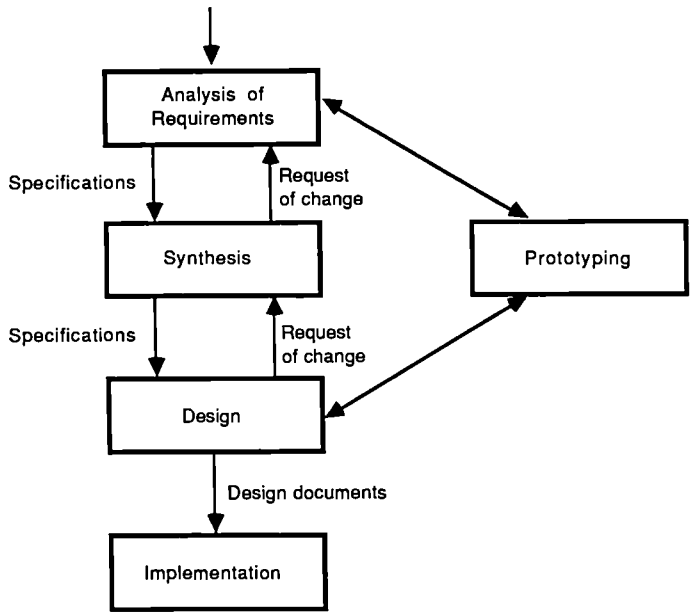


Figure 2

3. The Knowledge-based Assistants

Each of the Knowledge-Based Assistants will include knowledge about both the method the developer has to follow in the Analysis or Design stages and the particular application area of the system to be developed. We call the former Methodical Knowledge and the latter Domain Knowledge. In such way the Assistants can provide the user with general suggestions and checks about the various steps of the methods, the criteria to be observed in each step and the heuristics useful to observe such criteria. Furthermore, more specific suggestions and checks related to the Analysis and the Design of systems in a given application area will be provided. The knowledge based assistants will give advice about the decision-making process, performing automatically some transformations (whenever possible), supporting the development and the retrieval of project documents, keeping track of the decisions for explanation purposes. A description of the Analysis Assistant in terms of its capabilities, its knowledge organization and representation is shown in the following sections.

3.1. The Analysis Assistant

In this section we try to explain what are going to be the main features of the Analysis Assistant. We describe them on the basis of the exploited kinds of knowledge. The Knowledge Base of the Analysis Assistant, as mentioned above, comprises mainly two kinds of knowledge: Methodical

Knowledge and Domain Knowledge. Within these two major subsets of knowledge we have established further classifications with the purpose of establishing its organization. The Methodical Knowledge consists of the rigorous laws of the DAFNE method and some "good criteria" or "well-established empirical expertise", independent of a particular application, which allow us to satisfy the method's laws. On the other hand, the current Domain Knowledge contains "well-established empirical expertise" to adhere to the method in a particular application area.

The Analysis Assistant will substantially provide the user with suggestions and checks related to the Analysis documents which are the basis of the two-way communication between the Assistant and the user. The suggestions and checks provided by the Analysis Assistant, if accepted by the user, will be transformed into editing operations on the Analysis documents.

3.1.1. Exploiting Methodical Knowledge.

A classification within the Methodical Knowledge is the following:

- Structure of the method;
- Criteria of the method;
- Domain-independent heuristics.

The structure of the method has to be represented in the Assistant in order to give to the user the possibility of asking about the various links connecting the stages of the Analysis method. The user following the method will always have the way of knowing the next method stage and the specific steps composing a stage. In other words the user must always have the chance of asking "What have I to do now?" or "What is the way for accomplishing this Analysis stage?".

When the user is carrying out a stage of the method, the Assistant will supply him/her with the criteria to be observed in the accomplishment of that stage. Actually these criteria are part of the DAFNE method, but they deal with the contents of the documents. Indeed their exact purpose is to tell the user what he/she has to describe in each single stage, from which viewpoint and which is the proper level of detail in order to have optimal (as complete and consistent as possible) specifications. Besides providing explicitly the criteria to the user, the Assistant will be able to apply them in order to verify the Methodical consistency among the already developed Analysis documents.

In some cases it is possible to have heuristics, coming from the expertise gained in following the Analysis method, which allow the analyst to satisfy the criteria of the method. The heuristics will allow a non-expert analyst to behave as an expert with respect to the development of Analysis documents.

3.1.2. Exploiting Domain Knowledge.

The heuristics mentioned in the previous section (those independent of the application area) are certainly not the only source of all the advice the Assistant is going to provide to the analyst. Indeed, the most useful heuristics are seldom independent of the application area. Usually they are more

effective if related to the concepts of a Domain [20]. For example, it is surely more convenient to have some alternative functional decomposition of the system at hand rather than general domain-independent criteria to do it. Even more useful is to have alternative decomposition on the basis of some parameters (e.g. non functional requirements). Functionally speaking, the Domain Knowledge can be viewed as an enhancement and a specialization of the Methodical Knowledge heuristics described in the previous section.

Furthermore, by exploiting the Domain Knowledge the Analysis Assistant will be able to verify, in some cases, the adequacy of the current Analysis documents to some general concepts of the Domain. Domain adequacy checks aim at advising the Analyst that the requirements he/she is specifying are not completely in accordance with the Domain criteria included in the Assistant's Knowledge Base.

Another functionality aiming at a complete exploitation of Domain Knowledge is the possibility of establishing synonyms. When the user is creating the Analysis documents representing the requirements of his system, he may like (or be forced) to use names different from the ones known by the Assistant. Leaving to the Analyst the chance of fixing synonyms between his own labels and the Assistant's ones has the effect of having user's labels connected with the right Domain concepts. In such way the Analyst will be able to access Domain information through the use of his own labels.

3.1.3. Editing operations.

The main goal of the current Analysis Assistant prototype is not to supply the user with sophisticated editing facilities for building the various Analysis documents, as they can suitably be supplied by syntax oriented tools. However it is unavoidable to provide some editing operations which will be invoked automatically by the Analysis Assistant as consequence of the dialogue with the user when he/she accepts certain Assistant's advice. Some documents can even be automatically built on the basis of other documents. So a tight integration between the proper Assistant and an editor component is necessary, in such a way that they share the same internal representation of the Analysis documents.

The ESPRIT Project GRASPIN has developed a sophisticated syntax and semantics driven editor dealing also with graphical languages [21] and an integration of Aspis with such tool might be a goal for the future work.

3.2. Knowledge organization in the Analysis Assistant

As we have pointed out in the previous section, while Methodical Knowledge refers mainly to the knowledge about the DAFNE laws and stages to be developed, Domain Knowledge concerns about specific application fields. The main goal the Analysis Assistant has to achieve, from the user point of view, is to provide domain-dependent suggestions and advice while he/she is carrying out a particular stage of the method. So, the knowledge has to be used and organized at two different levels of abstraction in such a way that Domain Knowledge is seen through Methodical Knowledge.

A domain rule or fact will be included and exploited by the Analysis Assistant only in relation to a specific methodical step.

Semantic networks with production systems attached as attributes to nodes are a suitable solution for representing all the needed knowledge. The classes of the semantic network are going to describe the abstract form of the set of objects and define the type of their attributes. Classes can be structured in a hierarchical way by means of the IS_A relation. Production systems, in general, are used for representing the procedural knowledge (e.g. the way of checking whether domain or methodical criteria are satisfied).

3.2.1. Representing the documents.

Every SA Model is a tree of diagrams each of which containing boxes and arrows. The representation of a Model, by using Semantic Networks, is quite straightforward. Each component of the Model (diagrams), and each component of diagrams (boxes and arrows), are represented as a separate objects. Such objects will thus be properly linked together in order to reflect the SA Model structure. The classes defining the shape of the objects and the names of the links among them are shown in figure 3.

The tree structure among diagrams is obtained by means of the "refinement" and "boxlist" links which allow one to state that a box is both a component of a diagram ("boxlist") and the parent of another diagram (its refinement).

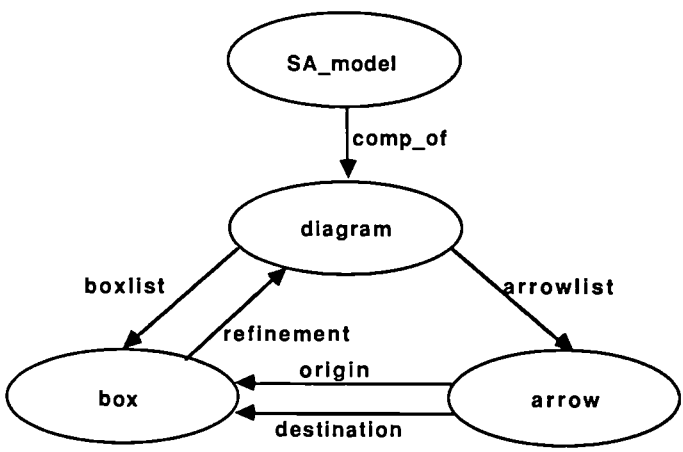


Figure 3

In the overall system, SA Models are defined as a sort of abstract data type. The set of operations and rules for manipulating all the SA diagrams included in a model, is included in the SA Model class (such operations are just the editing facilities we have mentioned in the previous chapter). The editing operations will be made available where necessary, by exploiting the network inheritance features. Every methodical stage whose purpose is to create an SA model will be connected by an IS-A link to the SA-Model class and will inherit all the related operations.

3.2.2. Representing Methodical and Domain Knowledge.

In order to represent Methodical and Domain Knowledge using the semantic network, we are going to have a class in the network for each stage of the Analysis method and each concept of the Domain at hand. Each of these classes can have production systems as attributes. Such production systems, which are going to represent the most important chunks of knowledge that a Methodical stage or Domain concept can have, may be activated either by the user or automatically by the Assistant itself.

The classes representing method stages are linked together on the basis of the sequence of the method steps and the ones representing domain on the basis of the Domain relations occurring among Domain concepts. Moreover, as we have commented in the previous section, Domain Knowledge can be considered as a specialization of Methodical Knowledge. So, the classes representing Domain Knowledge will be connected via IS_A links to the related Methodical Knowledge classes. Figure 4 shows the network representing a subset of the method stages with an example of a Domain class that is a specialization (IS_A) of the Model_of_the_Environment class.

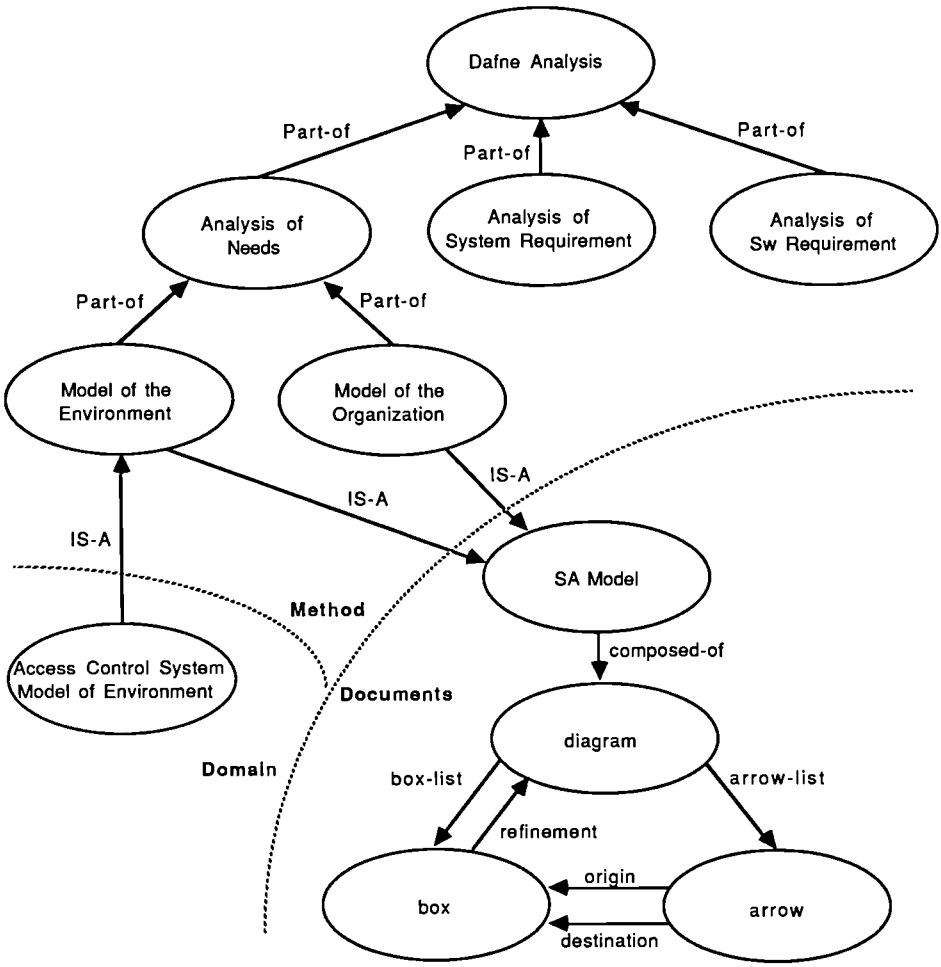


Figure 4

The Analysis Assistant, having the purpose of exploiting the knowledge contained in these classes, will thus navigate through the method classes according to their links focussing at each moment on one specific class representing a method step. When the "focus" is on a given method class, the Analysis Assistant will provide the user with a list of predefined operations associated to that class, which in general include the possibility of obtaining suggestions or consistency checks and also to exploit the knowledge contained in the related domain class. Such Domain Knowledge can be exploited by focussing the Domain classes which are the specialization (IS_A link) of the current method step.

From the user point of view, indeed, the inheritance mechanism and the focus movement will be transparent and he will only see a greater set of feasible operations and requests when he requires to exploit also the Domain Knowledge. So there will be two major levels of abstraction in the user

interaction and the analyst will have the possibility of deciding the proper level and the kind (Methodical and/or Domain) of help he needs.

4. RSLogic: The Rationale

The specification of a system has to take into account several issues: typically, the kind of data accepted, manipulated and produced by the system and the operations on those data. Indeed the specifications must point out the structural and the functional aspects of the system. RSlogic provides a set of primitives for dealing with the above issues. Moreover, the specifications have to deal with dynamic situations. In fact, there is a notion of passing of time: certain activities of the system under specification occur within specific time constraints, others occur if some time constraints are not satisfied. Dealing with the time issues introduces new problems.
In the literature one can find a lot of methods for dealing with the time issues. Typically most of these methods are based on temporal logic [22,23]. In fact the ability of temporal logic of expressing the relationships among the states of the system has a simple temporal interpretation: states have to be understood as *points of time* and the relation among states is the causal/temporal dependency relation. Using the temporal operators one can specify properties in terms of the states of the system, properties that sound like *eventually* or *necessary* and so on, without considering time explicitly. We believe that the simple temporal interpretation described above is not powerful enough to express some timing constraints. The crucial point is that we do not only want to specify problems in which events eventually happen, but also problems in which events must happen in some fixed time. Moreover time constraints are closely related to the functionalities to specify, since they have start and end, and the time passing is to be modeled together with the functionality. In order to achieve these goals RSLogic has an explicit notion of time; from this point of view it is closely related to [24,25,26] see [27] for a review.

The basic concept to understand is the concept of *position*. Positions denote places of the specified system: a system is specified by describing its positions. Because we want to state the rules for specifying how data flow in the system under specification dealing with both the structural, functional and timing constraints, we associate to a single position its data and the time the position has held its data. The above features lead to the concept of *event*. An event is a triples <q, v, t>, where q is a position, v is a data and t a time stamp. A triple <q, v, t> denotes that the position q holds the data v at time t. A *state* is a set of events, called *event set*, with a *state actual time*. Time stamps in the event set can be either less or greater than the actual time: a time stamp t less than the actual time means that the value v reached the position q at time t; a time stamp greater than the actual time means that the value v will reach q at time t. In other words, the state is to be understood as the observation of the *active positions* i.e. the positions which either hold data or are going to hold data at a given time. States are then organized in *histories*. A history is a sequence of states possible for the system under specification. The set of valid histories, i.e. the sequences of states that can be effectively stepped, is called the set of *Viable Histories*. The notion of viability is both a structural and a semantic notion because it reflects the structure of the system, but, at the same time, it depends on the properties of the data. For instance, a sequence of positions may be viable for some data and not for others. In RSLogic the collection of viable histories is defined by a set of axioms, called *Transition Rules*. A transition rule states that the presence of data, with suitable properties, in some positions can produce other data in other positions, within given intervals of time. The transition

rules of RSLogic describe the ways a state of the system may change. In this way the set of possible viable histories is defined. Thus the behaviour of the system is understood as a legal sequence of states, namely a viable history.

The amalgamation between SA and RSLogic is straightforward. RSLogic positions correspond to positions in the diagram. Typical examples of positions are input and output arrows of RSLogic. As a consequence, Viable Histories correspond to paths in the diagram.

4.1 RSLogic Syntax

Specifications involve the description of the data accepted, manipulated and produced by the system. In our logical setting, this fact is reflected in the syntax by the introduction of a Data Alphabet and expressions.

Data Constants	(c)
Data Variables	(x)
Data Functions	(f)
Data Expressions	(exp)

$exp ::= c \mid x \mid f(exp_1, \ldots\ldots, exp_n) \mid$
$\quad$ if $p_1(exp_1, \ldots, expn), \ldots., p_h(exp_1, \ldots, exp_k)$ then exp else exp

In order to describe the temporal features of the system at hand, a time alphabet and expressions are introduced.

Time Constants	(T)
Time Variables	(t)
Time Expressions	(texp)

$texp ::= T \mid t \mid texp_1 + texp_2 \mid texp_1 - texp_2$

Now we introduce the distinguishing feature of our logical formalism: the histories.

Position Constants	(q)
Event	(e)
Event Set	(es)
Event Set Variables	(s)
State	(st)
History Variables	(h)
Histories	(hs)

$e ::= \langle q, exp, texp \rangle$
$es ::= \{e_1, \ldots\ldots\ldots, e_f\} \mid s \mid \{e_1, \ldots\ldots\ldots, e_f\} \cup s$
$st ::= \langle es, texp \rangle$
$hs ::= h \mid st \mid h_1 \| h_2$

where || denotes the concatenation of histories.

Finally, transition rules, data axioms and queries are used for the specification and prototyping the system.

Data Atomic Formulae	(daf)
Time Atomic Formulae	(taf)
Viable History Facts	(hf)
Data Rules	(dr)
Transition rules	(Trs)
Queries	(qr)

daf ::= $p(exp_1,.....,exp_n)$
taf ::= $exp_1 = texp_2 \mid texp_1 > texp_2 \mid texp_1 < texp_2$
hf ::= **Viab**(hs)
dr ::= daf :- $daf_1,.....,daf_n$

Trs ::= **From** $<q_1,exp_1,texp_1>, ... ,<q_n,exp_n,texp_n>$
and not-occur $q'_1,....,q'_r$
with $daf_1,...,daf_h,taf_1,....,taf_k$
Cons $q"_1,.....,q"_v$
Produce $<q"'_1,exp"'_1,texp"'_1>, ... ,<q"'_m,exp"'_m,texp"'_m>$

qr ::= **Viab**(hs) **with** $daf_1,...,daf_n,taf_1,....,taf_m$

4.2 RSLogic Specifications

Data rules allow to state the properties of data involved in the system. Time rules state relations among the time constants used in the system description. They have exactly the form of Prolog rules and facts. The time stamps in the **Produce** part are assumed as relative to the time of application of the rule. For instance:

From <q,v,t> ..... **Produce** <q',f(v),T>

means that a new value f(v) will be produced in q' at time t+T.

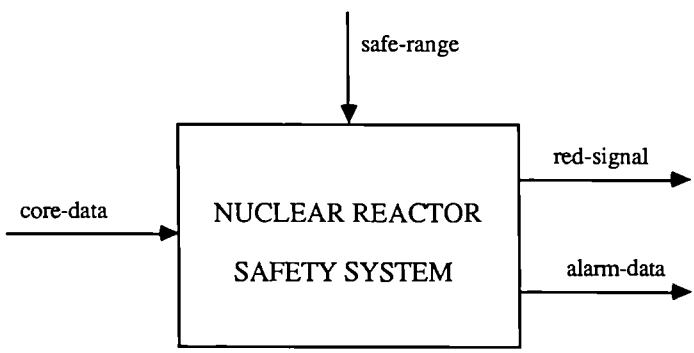
The idea is that transitions transform states into states and, consequently, develop the histories. Any transition, in general, adds new data in some new positions and consumes data in some old positions. The informal meaning of a transition rule is the following

if data, with given properties are present in some given positions, and other given positions do not hold data, then new data are computed in other given positions, each in a given (possibly different) interval of time.

The application of a transition rule produces an event set, in which different events may have different time stamps. Since time stamps may be different, it is not immediate to determine the time stamp for an event set as a whole and the application time for a transition rule. The time stamp of a whole event set, whose computation is explained below, is directly associated to it in order to form a state.

It is quite natural to assign to a transition rule application the time at which all the conditions for the rule become verified. If the rule does not require the absence of data, by **not-occur** specifications, the situation is straightforward: the time assigned to a transition in a state s is the maximun time in the subset of events (i.e. positions) necessary for the transition. If the rule requires the absence of data in some given positions, the matter is a little less obvious to treat. Indeed , it is possible that the rule is triggered by the disappearence of some data as effect of a previous transition. But the time of disappearence of data is not registered in the actual event set. This is the principal motivation for the explicit occurrence of a time stamp as time of the whole state.The basic idea is that the time assigned to the application of a transition rule, in a given state, is the maximum time of the subset of the necessary events of the state and the time stamp of the state itself. The same time stamp is assigned to the produced state.

Notice that, the time assigned to a state is the time in which the generating transition fires and not the time in which the effects of the transition occurr. Usually a state contains events with a time stamp greater than the time stamp of the state itself. Consider the following example. The inputs of a safety system (e.g. for a nuclear reactor) are a set of safe ranges that must be observed by data coming from the reactor itself (core data). Suppose that, whenever core data are out of the safety range a red signal is immediately set on and some more detailed alarm data are output after, say, a few seconds.



Consider the following history fragment:

$$<\{<\text{safe-range},v_0,T_0>,<\text{core-data},v_1,T_a>\},T_1> \parallel$$
$$<\{<\text{safe-range},v_0,T_0>,<\text{red-signal,on},T_a>,<\text{alarm-data,d},T_2>\},T_a>.$$

The presence of data v_1 out of safety ranges in position core-data has produced the addition of the two events <red-signal,on,T_a> and <alarm-data,d,T_2> with $T_2 > T_a$. The time assigned to the new

state is T_a, since this is the time stamp of the event that caused the transition. The event <alarm-data,d,T_2> may be thought of as a future event, which is at now precisely foreseeable.

5. The Support Assistant

The Support Assistant (Prototyper and the Reuse) are characterized by the fact that they depend on the availability of formal specifications and designs. Their functionalities are requested by the user during a session with a Knowledge-based Assistant whenever necessary, so they have to be tightly connected with the other assistants.

The Prototyper Assistant is in charge of executing the formal specifications, namely the RSL annotations of the SA diagrams, in order to animate the system under development and to provide the basis for the various consistency checks. Currently a tool which executes the RSL transition rules is under development and its integration with the Analysis Assistant is planned for the first ASPIS prototype.

The animation environment for RSLogic has been built on top of Prolog for obvious reasons. Indeed, part of the specifications, e.g. preconditions are Prolog goals. The animation environment consists of a Translator, an Executor and an Interface.
The translator takes the RSLogic transition axioms, the function definitions and the queries as inputs and yields a collection of Prolog clauses which are interpreted by the Executor. The generated clauses contain calls to predicates defined in the Executor, e.g. the predicates which implement the unification algorithm over sets and the predicates necessary for the execution of functions. Indeed, Prolog does not allow the use of functions and it performs only syntactic unification
One of the main components of the Executor is the conflict resolution module. This module determines the axiom which can be applied in the current step, performing some computations of temporal relations over time symbolic constants. Such a computation can require interaction with the user who is, possibly, asked to choose among different alternatives.

The verification of the preconditions occurring in the transition axioms is left to the underlying Prolog executor. The interactions between the Translator and the Executor are made transparent to the user by the Interface module. The Interface allows the user to interact with the system via the use of menus.

The Reuse of old specifications, designs and code on large scale and in a structured way is another important field of current research. Mainly two problems will be tackled in the future work on the Reuse Assistant: to access to the various components and to verify semantically that they are the right objects. For the first task an informal approach by keyword searching is followed and for the second a more formal one by executing the related specifications in order to check whether they present some properties or not and to understand their semantic.

5. Conclusions

The ASPIS project tackles several problems which are considered to be crucial in the SLC, such as the definition of formal specification languages, rapid prototyping, software reuse and the

development of knowledge-based tools, which embody both knowledge about a methodology and knowledge about the application domain. Most of the investigation work on the application of AI techniques to Software Engineering fields has been completed. Now the project is going to reach an important milestone: the first prototype. The intent is to show that our ideas can work and that our apporach can improve the efficiency of software development and enforce the semantic consistency of the software results as they evolve from the specification to the detailed design.

Acknowledgments

We should like to thank all colleagues of the ASPIS team for the useful discussions we have had with them.

REFERENCES

[1] *R. Balzer, C. Green, T. Cheatham*, "Software Technology in the 1990's Using a New Paradigm", **Computer**, IEEE, Nov. 1983.

[2] "Special Issue on Artificial Intelligence and Software Engineering", **IEEE Transaction on Software Engineering**, SE-11, Nov. 1985

[3] ASP/40: "Specification and Design of the Analysis Assistant", Project Deliverable, Mar. 1987.

[4] ASP/38: "RSLogic", Project Deliverable, Mar. 1987.

[5] ASP/39: "Knowledge Representation System", Project Deliverable, Mar. 1987.

[6] ASP/32: "The ASPIS Rule-Based Knowledge Representation Language", Project Deliverable, Nov. 1986.

[7] *R. Balzer et al.*, "Operational Specification as basis for Specification Validation", in **Theory and Practise of Software Technology**, Ferrari, Bolognani, Goguen Eds. North Holland 1983.

[8] *P. P. Chen et al.*, "Formal Specification and Verification of Distributed System", Specification Validation", in **Theory and practise of Software Technology,** Ferrari, Bolognani, Goguen Eds. North Holland 1983.

[9] *P. Zave*, "An operational approach to requirements specification for embedded systems" **IEEE Trans. on Soft. Eng.**, SE-8, 1982

[10] *G. Lojacono*, "DAFNE: Analysis", Italsiel Internal Report, 1984.

[11] *D. T. Ross*, "Structured Analysis (SA): a Language for Communicating Ideas", **IEEE Trans. on Soft. Eng.**, SE-3, Jan 1977.

[12] *D. T. Ross, K. E. Schoman*, "Structured Analysis for Requirements Definition", **IEEE Trans. on Soft. Eng.**, SE-3, Jan 1977.

[13] *D. T. Ross*, "Applications and extensions of SADT", **IEEE Computer**, 1985

[14] *P. P. Chen*, "The Entity-Relationship Model -Toward a Unified View of Data", **ACM Trans. on Data Base System**, Vol. 1, No. 1, March 1976.

[15] *G. Pacini, F. Turini,* "Animation of Software Requirements", **Industrial Software Technology**, (R.J. Mitchell Ed.), P. Peregrinus Ltd, London, 1987.

[16] *M. Degl'Innocenti, G. Ferrari, G. Pacini, F. Turini*, "RSLogic: A Formalism for Requirement Specification and Animation", (submitted for pubblication).

[17] ASP/41: "Specification of the Design Assistant", Project Deliverable, Mar. 1987.

[18] *E. Yourdon, L. Constantine*, "Structured Design", Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1976.

[19] ASP/42: "Specification of the Syntesis Process", Project Deliverable, Mar. 1987.

[20] *B. Adelson, E. Soloway*, "The Role of Domain Experience in Software Design", **IEEE Transaction on Software Engineering**, SE-11, Nov. 1985.

[21] GRA/80: "Architecture of the Final GRASPIN Workstation Prototype", Project Deliverable, Oct. 1986.

[22] *L. Lamport* "What Good is Temporal Logic", Proc. **IFIP**, (R. Mason Ed.), North Holland, 1983.

[23] *A; Pnueli* "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends", in **Current Trends in Concurrency** (J. de Bakker Ed.) LNCS 224, 1986

[24] J. *Allen, J. Koomen*, "Planning using temporal logic" **IJCAI 83**, 1983

[25] *R. Lee, H. Coelho, J.C. Cotta*, "Temporal Inferencing on administrative databases" **Information Systems**, Vol. 10, N. 2, 1985

[26] *R. Kowalski, M. Sergot*, "A logic based calculus of events" **New Generation Computing**, Feb.1986

[27] *F. Sadri*, "Representing and reasoning about time and events: 3 recent approaches" Internal Report, Imperial College, 1986

Project No. 973

Integrating Graphics into Prolog

Nicola Preston

CRIL
12 bis, rue Jean Jaures
92807 PUTEAUX
France

The work described in this paper is partly funded by the European ESPRIT program (project p973)

1. Presentation of the Esprit project ALPES - p973

The Esprit project ALPES - p973 (Advanced Logical Programming EnvironmentS) aims to build the prototype of a high-level programming environment for logic programming and the Prolog language in particular. This 3-year project began in June 1986 and is based on the results of a pilot phase Esprit project : ALPES - p363. The specification phase ends with the first project deliverables in September 1987.

The aim of the project is to combine the efforts of several industrial and academic research groups in order to improve the usability and efficiency of Logic Programming languages. Despite its enormous potential, Prolog has seldom been used for many real large scale applications due to certain universally recognised shortcomings.

Twelve different issues are to be studied and prototyped in different 'Tasks' of the project before being integrated in a unique environment. The author has been working with M.J. Prospero of Universidade Nova de Lisboa and T. Gandilhon of BULL AI Research Centre in France on the Task 'Prolog and Graphic Systems', and this paper includes descriptions of the work and ideas of all three.

2. Why graphics extensions to Prolog ?

The only way for an interactive Prolog program to communicate with the end user in the traditional Edinburgh syntax is by the built-in primitives for text and line-feed input and output. This is very unsatisfactory, as dialogue with any interactive program is often made easier and more attractive to the user by a graphical presentation. In particular, user interfaces based on overlapping windows, menus, icons and a mouse pointing device have become so popular that their layout and functionality is almost standard. The ALPES programming environment will present such an interface to the Prolog applications programmer and he or she should in turn be able to write Prolog programs that present data in different windows and allow the end-user to choose different options with a mouse. Indeed, Prolog's suitability for rapid-prototyping means it could be used in the development of user interfaces, which involves experimenting with different configurations to find which is most ergonomic [1]. A graphical presentation of data structures and relations in the form of trees and other graphs is very helpful and is needed for various tools of the ALPES environment such as the debugger and the browser. Another use of graphics extensions might be to dynamically illustrate the behaviour of simulation programs or games-playing programs.

But as well as offering the possibility of user-friendly interfaces to any applications program, graphics facilities in Prolog also open the way for inherently graphical applications such as CAD and graphics databases. These fields call for accurate and complicated drawings and have traditionally used graphics programs employing a very different programming style from that encouraged by Prolog, being very specialised, procedural and algorithmic and not at all 'intelligent' in the sense that different representations of the same data are necessary for different purposes [2,3]. The potential contribution of Logic Programming to this field, which is now beginning to interest many researchers (see papers from recent Eurographics and IFIP working group 5.2 conferences) cannot be evaluated without graphics extensions to Prolog which allow the Prolog programmer to represent and reason about geometric

models of real world objects with the aid of drawings displayed on the screen.

3. Criteria

What are the criteria invoved in designing the integration of graphics facilities into Prolog in the context of the ALPES project ?

Already in the above paragraph the principal difficulty of defining graphical extensions to Prolog for the ALPES environment can be seen : the extensions must be general enough to serve the needs of writers of inherently textual applications who want a user-friendly interface and graphics programmers who want to define complex drawings. For the latter, there is no point in providing graphics facilities in Prolog if they inhibit the Logic Programming style of programming. The aim of the ALPES project is to encourage the use of Logic Programming, which is often rejected as being unsuitable for the development of real applications, so the graphics must be reasonably fast. This implies an interface to routines written in a lower-level language where necessary. Also, the ALPES project does not cover the programming from scratch of low-level graphics routines, which has already been done elsewhere and would consume too much manpower, so we must make as much use as possible of packaged software.

I develop the implications of these criteria in the following paragraphs.

3.1. Window manager interface

We must provide the means for the Prolog programmer to manipulate windows and, as we must interface to existing software as much as possible, this means interfacing to a window-manager. However, window-managers were not designed to manipulate accurate drawings of geometric models and have an approach and representation of images which is fundamentally different from that of the computer graphics software traditionally used for this application. We must therefore also interface to more traditional graphics routines whose output will be displayed in windows managed by the window-manager.

Window managers are usually programmed using the object-oriented programming paradigm and windows can only be manipulated using the very procedural functions which they have predefined as 'methods'. An interesting proposal for implementing a window manager in Prolog is given in [1] but we consider the interface to an existing window manager the least interesting of the graphics extensions from the point of view of Logic Programming and do not have the space to discuss it further in this paper.

3.2. Logic programming style

A pure logic program has both a declarative and an operational semantics. Conditional definitions or 'rules', shared variables, structured terms and non-determinism give it great expressive power. Prolog already has many non-logical features but graphics extensions should be integrated in such a way as to enable the Prolog programmer to come as close as possible to the ideal of a pure logic program, so as to benefit from its power and clarity, when manipulating graphical objects. In the following paragraphs I discuss in more detail how this can be achieved.

3.2.1. Declarative representation of drawings

Drawings should be represented in Prolog by declarative specifications which can be reasoned about like any other Prolog object and which can be interpreted to display the drawing on the screen. Relations between a drawing and the Prolog object which it illustrates can then be specified so that whatever the properties of the object they are passed to and automatically illustrated by its drawing.

3.2.2. Non-determinate drawing function

The action of drawing on the screen has no logical, declarative semantics in Prolog and must be implemented as a side effect. In order for the drawing of an object to be integrated into non-determinate Prolog programs as cleanly as possible, the side effect should be undone on backtracking. This is explained further below. The problem with undoing drawing on backtracking is that the

implementation becomes quite complicated if the drawing is superimposed on another drawing on the screen, which must be restored when the former is erased.

3.2.3. Drawings which change

Prolog has no notion of editing or changing the attributes of an 'object' except by successive instantiations during the resolution of a goal. Otherwise, for Prolog, once the value of an attribute of an 'object' changes, it is no longer exactly the same 'object'. A Prolog goal might include the definition of an object which is drawn on the screen. When the goal subsequently fails and backtracks to try another solution the drawing is erased on backtracking and redrawn according to the next solution. The undoing of side-effects on backtracking enables the dynamic illustration of the solution of a Prolog goal and like this a changing drawing is represented declaratively and cleanly.

3.2.4. Structure of drawings

Another way in which a drawing might be required to change for a given application is by the addition or deletion of parts.

The most natural way to describe most drawings is as composite objects made up of elementary parts, eg: a drawing of a car composed of drawings of wheels, windows, doors etc, where the properties of the drawing, such as colour, might be local to a part or apply to the whole drawing. It should be possible to build up the description of a drawing as composed of predefined 'building block' parts which are transformed or given different attributes. It must also be possible to describe several independent drawings superimposed or otherwise combined to form an overall picture, eg: in a chess-playing program the chess pieces must be drawn superimposed on the board [4,5].

If a drawing is structured into different parts it should be possible to display the parts individually, in particular to display parts as they become instantiated or to modify parts which are reinstantiated without having to erase and redraw the whole drawing (although those parts which lie on top of or underneath the modified parts may have to be redrawn).

3.2.5. Uninstantiated variables

A drawing can be represented as a term in Prolog which is interpreted by graphics routines to show the drawing on the screen. The drawing cannot be shown on the screen, however, until 'ground' (fully instantiated). Thus, if there are uninstantiated variables in the definition of a drawing given by a Prolog clause, it can only actually be drawn on the screen when all the variables have been fully instantiated by the Prolog resolution.

3.2.6. Drawing output

If a built-in predicate must be explicitly invoked for the drawing side-effect then the programmer has the choice of either showing successive alternative definitions of the object or only the final solution and different parts can be drawn and erased selectively. The alternative to using built-in predicates with graphical side-effects is to have the Prolog interpreter carry out the side-effects automatically at certain points, eg: showing the drawing on screen on the 'assertion' of its definition into a 'graphical database' or at the point at which the definition becomes fully instantiated, but it is difficult to define a scheme like this that is general enough for all possible applications.

3.2.7. Invertibility

It must be possible to translate the representation of a drawing in the graphics standard back into its representation in Prolog. Like this, tools such as graphics editors written in other languages and handling drawings represented as in the graphics standard can be used to generate drawing definitions which can be translated into terms to be used in Prolog programs. It is thus not necessary to write graphics tools in Prolog unless they make use of Prolog's capabilities.

3.3. Graphics standards

As explained above, we need to interface to exisiting graphics routines. Several graphics programming standards have been established which offer the benefits of portability and presumably a complete definition of the necessary functionality in the domains of application they were designed for.

Most graphics standards, being based on traditional programming styles, are extremely procedural. Before anything can be drawn on the screen successive levels of the graphics system must be initialised. The appearance of each output primitive depends on what has been drawn beforehand and what attribute values have been set beforehand in a completely free and unstructured way. Although obliged to interface to such graphics systems as explained above we will try to hide as much of this as possible from Prolog in order to enable a declarative programming style. The Prolog Graphics interface will therefore not be a Prolog binding of a graphics standard as we consider that this needlessly introduces a low and procedural programming level into Prolog [6,7].

4. PHIGS

We have decided to base the representation of drawings for the graphics interface on the proposed graphics standard PHIGS [8].

In the context of Prolog, PHIGS is very interesting because in its Centralised Structure Store (CSS) it holds a dynamic structured representation of all drawings. This representation is almost completely declarative and is traversed by the **post** function to show a drawing on the screen. If the definition in the CSS of a drawing (called a **structure**) is changed while the drawing is visible on screen (**posted**), then PHIGS automatically updates the drawing on the screen. This means that PHIGS routines can be used to implement backtracking graphics functions in Prolog.

Another interesting feature of PHIGS is that **structures** may invoke other structures as parts of themselves. If a structure contains an **execute structure** element for another structure then the second structure is invoked as a part of the first. The second structure may have its own structures-as-parts, and so on, so that a drawing may be defined by a tree or more complicated network of structures rooted in the single structure that carries the name of the drawing. The same structure can be invoked by many different parent structures. The structure definition is not duplicated in each invocation but pointed to. This means that if its definition is changed, it changes in all its invocations in all structures, and it will be redrawn according to the new definition everywhere where it appears on the screen at the time. This is very useful for CAD applications where the same basic part, such as a nut, may be used throughout different products.

A structure consists of a list of elements, which may be **output primitives** (eg: line, polygon, text), invocations of other structures (ie : **execute structure** element), **attribute values** (eg: linestyle, font) or other properties such translation, scale and rotation transformations.

A structure invoked as part of a parent structure inherits the **attribute values** of the parent as defaults which are overwritten by any attribute values of its own during traversal. An **output primitive** will be transformed according to the current **composite transformation**, composed from the **global** and **local transformations**. The global transformation is the composite transformation, inherited from the parent structure unless overwritten by a **set global transformation** element of its own.

Structure definitions are either loaded from files or created by performing incremental edit operations on the contents of an initially empty named structure. The structure contents can then be further edited at any time. If a structure which has not been defined is invoked by name by another structure then an empty structure with that name is automatically created and will be the basis of subsequent edit operations. An empty structure can be posted, in its own right or as part of another structure. Subsequent definition of the structure, like any edits to a posted structure will be visible straight away on screen, everywhere that the structure appears.

A named but empty structure is thus like an uninstantiated variable in Prolog. When it is defined by a PHIGS edit it is fully 'instantiated' both in the CSS and on screen. However, in Prolog partial instantiation of a variable is possible and could be a way of adding elements to a drawing but in PHIGS once a structure contains a single element it is considered fully defined and must be

5. Description of Graphics Facilities in Prolog

This is a description of the implementation decided upon according to the criteria discussed above, with the drawing representation and functions based on the PHIGS standard [9].

There will be a basic graphics interface which interfaces Prolog to functions defined in the PHIGS standard as well as to window manager functions by means of built-in predicates. This basic Prolog Graphics Interface is defined to ensure that as little as possible of the procedural and computationally heavy work that is better done in lower level languages is left in Prolog. It is minimal so as to be useful for a wide range of applications. By writing higher level predicates which call the built-in predicates of the Prolog Graphics Interface, by defining data-structures in Prolog and by writing meta-interpreters, we will then implement more specialised and powerful graphics facilities in Prolog.

Using various trial applications to test our ideas, we began by specifying the basic graphics interface. This is now well defined and I describe the main concepts below in some detail, and then go on to discuss higher-level graphics programming, giving examples of possible applications and programming aids.

5.1. Description of the Prolog Graphics Interface

Drawings will be represented in Prolog as structured terms. When fully instantiated, such definitions can be 'asserted' into the Prolog Graphics Interface by means of a built-in predicate. Another built-in predicate will be provided to traverse the definition and display on screen any drawing defined in the Prolog Graphics Interface.

Below I use **bold** for terms defined in PHIGS and *italics* for terms defined for the Prolog Graphics Interface.

5.1.1. Data structure

5.1.1.1. Image

An *image* is a fully-defined drawing in co-ordinate space, equivalent to a PHIGS **structure**. However, in the Prolog interface, in order to ease the analysis of drawings and make them more declarative, *images* are **structures** of a certain form. A **structure** is traversed from beginning to end so that an **attribute value** or other property applies to **output primitives** or **execute structure** elements occurring after it in the list only. Each **output primitive** or sub- **structure** of a **structure** may have different properties. But in an *image*, *properties* are separated from *parts* and the *properties* are applied to all the *parts* of the *image*. To give an **output primitive** different *properties* it must be defined as a separate *image* and invoked as a *part*. This means that it is always be possible to define a **structure** in terms of an *image*, but the *image* may have to be broken down into more part *images* than the **structure**. *Images* must belong to one of two *types* : *simple images* which contain only output primitives as *parts* and which are thus 'terminal' or 'leaves' in the *image* network, and *composite images* or 'nodes', whose *parts* are all invocations of other *images*. This image typing subsumes the difference in PHIGS between **local** and **global transformations** : the transformation *property* of a *composite image* is always inherited by its *parts* and pre-concatenated to any transformation *property* of the *parts*. An *image* is defined by : a name ; a *type* ; its *properties* ; its *parts*.

5.1.1.2. Scene

A *scene* is a collection of *images* to be *viewed* together in a window, possibly superimposed on each other. The *scene* is defined as a list of *image* names, each paired with a priority which gives the order of superposition. In PHIGS there is no declarative definition of a *scene*, which is implicitly built up by a series of **post** commands.

5.1.2. Built-in predicates

5.1.2.1. Definition and display of drawings

Three sorts of 'object' are 'created' and manipulated by the side-effects of the built-in predicates of the Prolog Graphics Interface : *images*, *scenes* and *views*. A *view* is a *scene* displayed on screen in a window. *Images* and *scenes* are objects in the database of the Prolog Graphics Interface only, but a *view* is also an object on screen : the contents of a window.

The built-in predicates which create 'objects' such as drawing definitions in the Prolog Graphics Interface will reject definitions that are not fully instantiated as the handling of uninstantiated terms is best done in Prolog. The predicates will fail or abort with an error message if the arguments are invalid. They will succeed as many times as their arguments can be 'matched' exactly like other Prolog predicates, except that on each re-try the side-effects of the previous try will be undone.

In the lower level graphics routines all names of 'objects' must be unique and fully instantiated. The Prolog Graphics Interface, however, will allow several objects to be given the same name and the 'matching' and instantiation of names will be as for other Prolog terms.

The built-in predicate *draw* defines an *image* in the Prolog Graphics Interface. If this *image* has been named as part of a *viewed scene* but not defined, then (as in PHIGS) *draw* causes the *image* to be displayed everywhere where it is *viewed*. Similarly, the predicate *scene* defines a *scene* and, in the case of *scenes* already named as *views*, causes the *scene* to be displayed in its window.

The predicate *view* displays a *scene* in a window, completely covering any previous contents of the window.

5.1.2.2. Inquiry predicates

If the definition of a drawing is implemented as the side-effect of a non-logical predicate then there must be a complementary predicate which gives the drawings already defined. Logical predicates with a declarative semantics are reversible but if we try to use the same predicate to 'assert' the definitions of drawings and to find which definitions have been 'asserted' then the semantics of the predicate are not clear when the arguments are bound variables. The built-in predicates *drawn*, *is_scene* and *viewed* will therefore give the definitions of 'objects' currently exisiting in the Prolog Graphics Interface.

5.1.2.3. Modification of drawings

The side-effects of the drawing definition and display predicates will be undone on backtracking, so image definitions can be handled in a completely declarative way in Prolog, and the programmer can place the display predicate to achieve the desired effect on screen. But for speed or aesthetics of display update, or where an incremental programming style is more natural, built-in predicates will also be provided to access the definitions of drawings and modify or delete them, with the side-effect that they are automatically updated wherever they are visible on screen. These predicates are defined to be close to PHIGS functions for maximum speed of display refresh.

5.1.2.3.1. Erasing definitions

Erase removes the definition of the named *image* from the Prolog Graphics Interface. If the *image* is part of a *viewed scene* then *erase* causes the *image* to be deleted from the screen everywhere where it appears.

Similarly, *erase_scene* removes the definition of the named *scene* from the Prolog Graphics Interface and causes any *views* of the *scene* to be erased from their windows.

5.1.2.3.2. Replace and redraw

Replace takes as arguments two *image* names, and replaces all invocations of the first *image* as a *part* by the second *image* name , and redraws the first *image* as the second everywhere where it is *viewed* as a *part*.

Redraw is similar to *draw* but if the named *image* is already defined in the Prolog Graphics Interface then, instead of adding another definition, it replaces the existing definition and redraws it everywhere where it is *viewed*.

5.1.2.3.3. Editing

replacepart
replaceprop
addpart
addprop
takepart
takeprop
add_element
take_element

For the *drawn image* or *scene* named, replace, add or delete a *part*, *property*, or element (*image*-priority pair in a *scene* definition) and update any views of the *image* or *scene*.

5.1.2.4. User-interaction

Unlike in PHIGS where **output primitives** are grouped into **classes** for these operations, they will work at the level of *images* in the Prolog Graphics Interface. Prolog pattern matching used on the *image* names defined in the application already gives the same functionality as PHIGS **classes**.

PHIGS allows synchronous and asynchronous models of user input but in the Prolog Graphics Interface it will be left synchronous, as it is now in Prolog, ie: moving or clicking the mouse will not have any effect until the user is explicitly asked to pick something on the screen. This is because asynchronous input would require the model of execution of Prolog to be completely changed.

Pick and configurable *menu* functions will be provided as built-in predicates so that the display of the menu, prompt, highlighting of the valid choices, erasing of the menu once the choice has been made, etc. can be left to the low-level graphics routines. Other built-in predicates will provide tracking and rubber-banding feedback for well-defined interactive functions, such as asking the user to fix the end-point of a line, opposite corner of a box, etc.

Built-in predicates will provide non-declarative *highlight* and *make_invisible* functions which operate on entire *images* and are considered to be 'temporary', ie: are not reflected in the Prolog Graphics Interface *image* definitions . *Highlighted* (ie: inverse video) and *invisible* could also be included as 'permanent' *properties* of *images*, to be intercepted by the Prolog Graphics Interface and implemented using the highlight and make_invisible functions.

5.2. Higher level graphics programming

5.2.1. Automatic display

Let us start by considering aids to automate the display function. A draw_and_view predicate could take an image definition and, using defaults for attributes not specified in the call, automatically define a scene containing just that image and display it in a window sized to be just big enough. The next step is to automatically display either all image definitions or all scene definitions in the Prolog Graphics Interface.

To automate drawing further, we can write a meta-interpreter for applications which consist of a graphical illustration of objects or events in a program which is not inherently graphical. For this, we try to imagine which types of Prolog object or event might need to be drawn. For example, certain clauses can be designated as having a graphical representation which is to be automatically displayed, either, for clauses to be displayed 'statically', when the clause is added to the Prolog database by a 'consult' or 'assert' or, for clauses to be displayed 'dynamically', when the clause is solved as part of the resolution. The picture of a clause will be erased again if the clause is respectively 'retracted' or backtracked over during resolution. All clauses depicted 'dynamically' during the resolution of a

particular user query can be erased when the user types carriage return after the resolution of the query. Sets of clauses, or events such as the beginning or end of a resolution could also be given representations.

In the following simple maze program the Prolog resolution of the path through the maze, including backtracking, can be dynamically illustrated using such a meta-interpreter. We designate the 'maze_fragment' unit clauses in the database which pre-define the maze as to be depicted on 'consult' and the clause 'path_fragment' each time it is solved. In order to draw an outline around the maze, which is not directly defined by the 'maze-fragments', we must define the graphical representation of the set of all maze-fragment clauses or an initialisation drawing to be displayed at the start of this user-query.

```
maze(End,End,Path).

maze(Current_Node,End,Path_so_far) :-

   path_fragment(Current_Node,Next_Node,Path_so_far),
   maze(Next_Node,End,[Current_Node|Path_so_far]).


path_fragment(Node1,Node2,Path_so_far) :- some_maze_fragment(Node1,Node2),
                                         NOT member(Path_so_far,Node2).


some_maze_fragment(Node1,Node2) :- maze_fragment(Node1,Node2).

some_maze_fragment(Node1,Node2) :- maze_fragment(Node2,Node1).
```

5.2.2. Redisplay

Different Prolog terms in a program may represent the same drawing on screen, and when such a term is displayed it must replace the current drawing. Alternatively, it may be the same Prolog term in different states of instantiation which represents a single changing drawing on screen.

The above example can erase unsuccessful path-fragments on backtracking because the drawings are independent. In a drawing of a tree where the nodes are of variable size, however, in order to fit in a new subtree substantial parts of the tree may have to be reformatted. Consider the following simple meta-interpreter to build and dynamically display the resolution tree, Tree, of Goal : <- and & are operators representing the mother-daughter and sister relationships in the tree originating from :- and , respectively in the program clauses.

```
anim(Goal,Tree) :- anim1(Goal,Tree,Tree).


anim1(true,true,Tree).

anim1((Next_subgoal,Rest_subgoals), Next_subtree & Rest_subtrees, Tree) :-

    anim1(Next_subgoal, Next_subtree, Tree),
    anim1(Rest_subgoals, Rest_subtrees, Tree).

anim1(Goal, Goal <- Subtree, Tree) :- special_redraw(Tree),
                                      clause(Goal,Subgoals),
                                      anim1(Subgoals, Subtree, Tree).
```

The term 'Tree' is gradually instantiated by the resolution and at each cycle of the meta-interpreter 'special_redraw' must find (solve) the format of (the instantiated part of) the Tree image and display the

new image in place of the old. If the image is broken down into parts so that it has the same structure as Tree, then changes in Tree as different clauses are tried and succeed or fail will be reflected as localised changes in its image. 'Special_redraw' could then compare each successive image of the whole tree with the current one and send the minimum number of edit commands to PHIGS for faster and more pleasing display update.

5.2.3. Uninstantiated variables in drawings

In the above example clause heads will be printed at each node. As the variables in these clause heads are instantiated by the resolution, it may be desirable to update the display accordingly. The meta-interpreter above can be modified to track changes in the instantiation of such variables and cause the display to be updated, although for complex terms this may be quite slow.

In the above example, Tree is partly instantiated but the image is always fully instantiated. There may be applications which require the definition and display of images containing uninstantiated variables representing parts which are to be displayed as soon as the variable is instantiated later in the resolution. On the instantiation of the variable, a sort of type checking must be carried out to ensure that the resulting drawing has a valid syntax.

5.2.4. Complex images

As it stands, the Prolog Graphics Interface described above is cumbersome to use to define or analyse complex drawings.

Inquiry predicates must be provided which give the properties of an image part resulting from inheritance, including the transformed co-ordinates, in abstract space and perhaps in the co-ordinate system of the view window.

The applications programmer will want to build-up and break-down complex drawings using prototype parts. An optional naming scheme can be defined which relieves the programmer of the burden of thinking up names for all images and can be the basis of tools to, for example, provide specialised versions of standard drawings with configurable properties, or provide copies of standard drawings which can be modified without modifying the standard. My ALPES project colleague, M.J. Prospero, has developed such a model.

5.2.5. Constraints and geometry

Template images will be provided for standard geometric shapes (square, triangle etc) and definitions of the geometric properties which constrain their co-ordinates. The programmer will want to manipulate these and other images which are partially instantiated, ie: with certain attributes which are variable, and governed by constraints on the values of these attributes.

Such constraints could be used, for example, in an circuit-board design application where the user suggests a place for a new component and the system checks constraints, such that certain types of components are never adjacent. Prolog is good at checking a suggested value against a set of constraints as in this example, and in the case where the programmer knows which quantity will be undefined, a procedure can be written to deduce its value from certain constraints as in tree formatting discussed above. However, a set of separately defined declarative constraints cannot necessarily be solved by conventional Prolog to give any one of several possible unknowns, because of Prolog's fixed search strategy. The solution of a set of mathematical constraints is entirely dependent on their order and conventional Prolog cannot solve simultaneous equations.

In order to reason about the basic geometric shapes mentioned above, an applications program would have to be equipped with knowledge of the different properties and ways of defining these shapes and the way shapes are related by transformations and by combination [10].

5.2.6. Object Oriented Programming

Drawing definitions in the Prolog Graphics Interface are hierarchies of objects (structures) with pre-defined inheritance of properties. A view of a drawing in a particular window is a separate object which inherits structure and properties from the definition. Furthermore, in most applications these

graphical objects will illustrate or represent other objects (abstract notions or models of real-life objects) in the Prolog program. It would probably be helpful to many applications programmers for the analysis, building, definition of relations between or operations on graphical and non-graphical objects to be able to view them using a similar paradigm to that of Object Oriented Programming [11].

6. Relation with other ALPES Tasks

From the discussion of higher-level graphics programming approaches above, it can be seen that some problems and needs arise which are not limited to graphics in Prolog. There is a need for extended unification, to attach side-effects to the instantiation of variables and impose type checking more efficiently than can be done by a meta-interpreter. We need to regroup the definitions of images and their properties, constraints and relations to other images or other Prolog objects and to be able to solve sets of constraints. There is thus a need for some sort of abstract data types or objects.

These questions come into the domain of other Tasks in the ALPES project where people are working on other extensions to Prolog, and in particular the work on Data Typing, Extended Unification, Functions and Objects. We are already co-operating in the Graphics Task with these Tasks and with the people working on the Editor, Browser and Debugger tools which need a graphical interface. In the next phase of the ALPES project this co-operation will be increased to arrive at an integrated and consistent programming environment.

References

[1] A. Michard, E. Monceyron, Le Systeme Graphique ASH-Prolog et son Utilisation pour le Prototypage Rapide d'Interfaces Homme-Machine, INRIA - Centre de Sophia Antipolis, 06560 VALBONNE, France

[2] I.C. Braid, Geometric Modelling, Notes Prepared for Eurographics '85 Tutorial.

[3] M.J. Pratt, Interactive Geometric Modelling for Integrated CAD/CAM, Advances in Computer Graphics 1, Ed. G. Enderle, M. Grave, F. Lillehagen, Springer Verlag, 1986

[4] R. Helm, K. Marriot, Declarative Graphics, 3rd International Conference on Logic Programming, London, 1986, Springer Verlag, 1986

[5] F. Pereira, Can Drawing be Liberated from the Von Neumann Style? Tech. Note 282, AI Center, SRI International, June 1983

[6] W. Hubner, Z.I. Markov, GKS Based Graphics Programming in PROLOG, Computer Graphics Forum 5 (1986) 41-50

[7] R. Krishnamurti, P. Sykes, A Graphics Standard for Prolog, The Prolog/GKS Binding, ESPRIT project ACORD task T3.1 Deliverable Report, January 1986

[8] Information Processing Systems - Computer Graphics - Programmers Hierarchical Interactive Graphics System (PHIGS), Draft Proposal ISO dp9592/1-198n(E), October 1986

[9] N. Preston, J.M. Prospero, T. Gandilhon, "Prolog and Graphics - Specification", Deliverable for WP3.1, ESPRIT project ALPES - p973, September 1987

[10] F. Arbab, J.M. Wing, Geometric Reasoning : A New Paradigm for Processing Geometric Information, International Symposium on New Direstions in Computing, Trondheim, Norway, 1985, IEEE Comp. Soc. Press, 1985

[11] T. Gandilhon, Proposition d'une Extension Objet Minimale pour Prolog, Programmation en Logique, Actes du Seminaire 1987, CNET

Project No. 892

DATABASE SOFTWARE DEVELOPMENT AS KNOWLEDGE BASE EVOLUTION

Matthias Jarke
Universität Passau
P.O.Box 2540, D-8390 Passau
F.R. Germany

Raf Venken
BIM S.A.I.N.V.
Kwikstraat 4, B-3078 Everberg
Belgium

Abstract. In ESPRIT project 892 (Development of Advanced Interactive Data-intensive Applications, DAIDA), we are investigating a knowledge base management systems approach to the development and maintenance of large interactive information systems. Individual development environments based on the conceptual systems modelling language SML, the design language TDL, and the database programming language DBPL are viewed as distributed knowledge bases, to be integrated via three facilities: prototyping in PROLOG, rule-based mapping assistants, and a global KBMS for documentation, communication and maintenance. This paper describes the DAIDA philosophy and architecture with a particular emphasis on the knowledge-based components, presents the status of a first implementation effort, and sketches potential areas of industrial impact.

1 INTRODUCTION

Database-intensive information systems are among the most important software applications. Due to their often extended lifespan (more than a decade is not unusual), their development and, in particular, maintenance have been an important bottleneck for the application of information technology in organizations. ESPRIT project DAIDA, now early in its second year, attempts to exploit the specific properties of this class of applications to build powerful database software development and maintenance environments, combining concepts from database design, artificial intelligence, and software engineering. This is in contradistinction to some other ESPRIT projects that strive for *general* software development support and therefore cannot utilize specific knowledge about the intended class of applications.

The goals of DAIDA can be described from two distinct perspectives within the ESPRIT work program. From the software technology viewpoint, DAIDA attempts to provide efficient knowledge-based support for a commercially important class of software applications, applying and extending results obtained in the area of formal specifications and software databases. Our main observation is that current formal methods do not suffice for a completely automatic transformation process in the area of programming-in-the-large; we therefore propose to encapsulate existing theory in several small-scale "expert systems" called language and mapping assistants, and to embed these mapping assistants in an environment (compatible with commercial systems such as PRADOS [RI85]) in which manual and automatic development tasks are integrated in a meaningful way.

The DAIDA team consists of: *BIM*, Everberg/ Belgium, prime contractor (Eric Meirlaen, Vera VanHeukelom, Raf Venken -- administrative manager); *Cretan Computer Institute*, Iraklion/ Greece (Alex Borgida, Maria Mamalakis, Manolis Marakakis, John Mylopoulos, Yannis Vassiliou); *GFI*, Paris/ France (Gerard Bonin, Alain Rouge); *Johann Wolfgang Goethe-Universität*, Frankfurt/ F.R.Germany (Gerhard Ritter, Joachim W. Schmidt, Martin Weigele, Ingrid Wetzel); *SCS Technische Automation und Systeme GmbH*, Hamburg/ F.R. Germany (John Gallagher, Rainer Haidan, Ingo Röpcke, Gernot Ullrich); *BP Research Centre*, Sunbury/ England (Horst Adler, Gerard O'Driscoll); *Universität Passau*, Passau/ F.R. Germany (Matthias Jarke -- technical manager, Manfred Jeusfeld, Thomas Rose).

From the Advanced Information Processing perspective, DAIDA proposes a new approach to the implementation of knowledge base management systems (KBMS) which extends ideas from coupling expert systems and databases [JV84]. Rather than just *accessing* existing databases as in the coupling approach, specialized information systems (with database and transaction structures) are *created* as backends to a knowledge representation language; this allows much more flexibility and efficiency than previous approaches, at the expense of a relatively high development cost for the KBMS. As the DAIDA environment improves, however, this expense may be considerably reduced.

In order to be able to concentrate on the novel aspects of the DAIDA architecture and to increase the industrial potential, it was decided to base the project on (variations of) existing tools and languages. In particular, BIM-PROLOG was chosen as the implementation language for most knowledge-based components. The other languages used in the project are adapted versions of the languages CML (developed in ESPRIT project LOKI), TAXIS (developed at the University of Toronto), and DBPL (developed at the University of Frankfurt). Experiments have been conducted with a number of commercial and ESPRIT-developed graphics tools, to be integrated into our development environment. One important aspect of the industrial potential of our work is portability, demonstrated in the project by the joint use of a SUN-UNIX and a VAX-VMS environment.

DAIDA is now at a stage where initial designs of the major components (the languages, their environments, the prototyping facilities, and the global KBMS) have been developed and implementation of the first of two intended prototype system has begun. The industrial partners expect to embed components similar to those developed in DAIDA into their commercial software development environments. This paper presents the DAIDA architecture and its rationale, and its potential impact on better and faster system development.

2. THE DAIDA ARCHITECTURE

2.1 A General KBMS Concept for Design Applications

A knowledge base management system (KBMS) for design applications can be understood as illustrated in figure 1. A group of developers interacts with a problem-solving environment which offers features such as interfaces, inference mechanisms, tools, design methodologies, and communication facilities. This environment (and through it also the developers themselves) has access to a common service, in figure 1 called the documentation knowledge base, in which persistent information about the design can be stored. Such persistent information should typically include:

- representation of completed and incomplete design objects in different versions and variants (the design *outcomes*),
- representation of the design history (as to communicate the design *decisions* and their rationales across time or among designers),
- representation of the methodologies, development standards, and special tools valid or available in the problem-solving environment (the design *rules*).

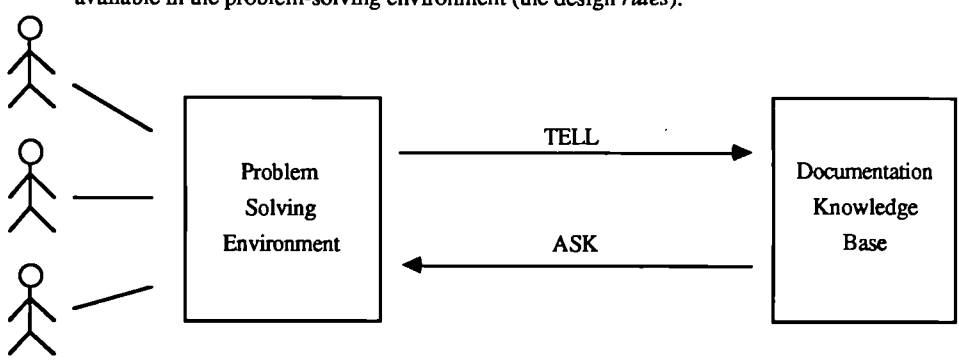


Fig. 1: Basic KBMS architecture

Conceptually speaking, a KBMS provides two functions [BL86]:

TELL : KB x Assertion --> KB´

maps a pair (consistent knowledge base, input) to a new consistent knowledge base. As implied by this definition, we would expect from a KBMS that its abstract TELL operation includes checking of syntactic correctness, consistency of the knowledge base and, in particular, integrity with respect to any design object types, design decision classes, and methodologies, development standards, and higher-level specifications. Obviously, the realization of the TELL operation is one of the main difficulties in creating a KBMS for design applications (and probably a major reason why a *generalized* KBMS does not exist yet). A traditional relational DBMS would be a simple KBMS where Assertions are only ground facts whose structure is predefined by a single initial, more powerful assertion (schema definition).

The other operation,

ASK : KB x Query --> Answer

allows the user to question the KB about its knowledge. Again, the implementation of such a mechanism for a design KBMS can be very complex, involving retrieval of complex objects, rule-based deduction or even general theorem-proving, representation switching, etc. By comparison, the query language in a traditional relational DBMS would just involve first-order predicates over the given simple structures.

2.2 Characteristics of Data-Intensive Information Systems

The main task in designing a KBMS can be seen as determining TELL and ASK operations that are representationally adequate to the task to be supported, yet can be efficiently implemented.To determine the special requirements for a KBMS architecture in DAIDA, we have identified two major distinguishing features of data-intensive information systems applications.

Firstly, databases have a double role. On the one hand, they store *beliefs* about the real world (e.g., beliefs about the properties of employees working in a company); on the other, they are becoming an important *part* of this real world (e.g., they are used by certain employees for particular tasks, where beliefs about these same employees may also be stored in the database). The dual role implies that the *validity* problems of data descriptions and the *longevity* of the data must be explicitly taken into account in a development and maintenance environment. In DAIDA, this is achieved by employing a systems modelling language, for describing a history of the subset of the world the intended database system is to model, as well as of the subset in which the system is to be embedded. For example, an existing database, for which an additional application is to be written, would be part of the latter subset. *Prototyping* in PROLOG facilitates the animation of these models for validation purposes.

Secondly, the *software world* prescribed as part of the above "world model" knowledge base, has the unique property that it can be *included into* the knowledge base; that is, the whole software development process can happen under full control of the computing environment (this distinguishes software CAD from, say, bridge CAD). Rule-based *mapping assistants* provide the DAIDA environment with a way to create (not just design) the software world semi-automatically by transformation decisions; two additional language levels, a declarative design language TDL (evolved from TAXIS [MBW80]) and a procedural database programming language (DBPL) address the specific problems of conceptual and program design. *Maintenance* can be partially automated by tracing the consequences of evolving beliefs about the combined external and software worlds. A *global KBMS* serves as a documentation and communication tool to coordinate development and maintenance work.

2.3 KBMS in DAIDA

Summarizing the last section, there are unique opportunities and challenges to be exploited in a database programming environment, due to the amount of knowledge we have about the applications (existing databases) and to the degree of control about our products; these opportunities and

challenges arise mostly in the areas of program transformation, design maintenance and prototyping. Thus, we can now derive a special case of the KBMS architecture: the general DAIDA architecture presented in figure 2.

The user community and the problem-solving environment are broken up according to the layers of world and system model, conceptual design, and database program design, and the transformations (mappings) among these. Similarly, the documentation KB takes the special form of a global KBMS. We now briefly describe the roles of these components (for a more detailed description, see [BORG87, JARK86]).

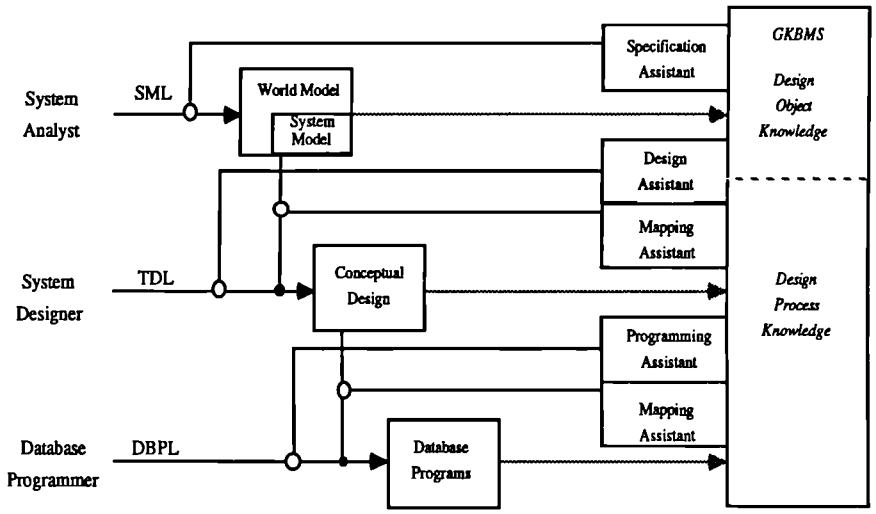


Fig. 2: DAIDA architecture

2.3.1 Conceptual World and Systems Modelling in SML

The highest level of the DAIDA architecture provides a conceptual model of the existing and intended systems and their environment. This "enterprise analysis" level has traditionally resisted formalization and was therefore often neglected, although it is known that errors at this level may have very severe consequences. The systems modelling language SML offers very flexible yet formal facilities to describe the history of a subset of the world (including the information system).

SML extends previous work on the requirements modelling language RML [GBM86] and the conceptual modelling language CML developed in ESPRIT project LOKI, by features specific to modelling systems objects and activities. It combines an object-oriented representational framework with a logic-based assertion language to describe constraints on objects and properties. Object classes are organized into generalization hierarchies with strict inheritance of properties and assertions. An important design consideration is extensibility of the language. To achieve this, classes are themselves considered objects which can be instances of metaclasses to be defined by the user. Similarly, property categories which describe common assertions on properties can be defined freely. An important metaclass in SML is SYSTEM_ACTIVITY which offers specialized property categories for describing information systems functions, based on SADT ideas [RS77]; the system objects are then related to the corresponding real-world objects. Finally, an interval-based time concept [ALLE83] allows the description not just of states but of histories of the real world. SML will extend the corresponding CML approach by integrating aspects of hypothetical future worlds and version concepts; these are needed to describe alternative possiblities for creating an information system.

The SML environment will offer a dialog management component, structure-oriented editors, and window-based interaction facilities for knowledge base manipulation and querying. The latter includes a prototyping facility which allows the animation of world and system model by example instances. The SML support system is being realized in BIM-PROLOG such that the implementation of the prototyping facility corresponds to the querying of properties.

2.3.2 Conceptual Data Design and Predicative Specification in TDL

While the first level of the DAIDA architecture describes the role of an information system in the world, the second level is responsible for the conceptual database design, interface facilities, and predicative specification. The design language TDL [BMMS87], derived from earlier work on TAXIS [MBW80], offers three major concepts for this purpose. As in TAXIS, *entity classes* offer means to describe a conceptual database structure (semantic data model) from which concrete database systems can be derived. *Transaction classes* represent abstract state transitions for this database, defined by predicative pre- and postconditions. *Script classes* describe the interaction of various subsystems with each other and with their users .

Supporting the central level in the architecture, TDL has to be designed in a manner compatible not only with the knowledge description language SML and the database programming language DBPL, but also with the prototyping language PROLOG. While it shares the object-oriented framework with SML, TDL's structure is much more rigidly focused on the systems design task. There are only predetermined metaclasses (entity, transaction, script, and exceptions) and property categories appropriate for these metaclasses. The general time concept of SML is replaced by a state transition mechanism with destructive update. Correspondingly, the representation of SML time information in TDL script and transaction definitions is the most central mapping task in going from the SML to the TDL level; it can build on substantial time-related research in AI and databases [ALLE83, SNOD86]. The predicative style distinguishes TDL from TAXIS; TDL is being designed in a way that predicative conditions can be easily derived from SML and mapped to DBPL; furthermore, at least a subset of them can be directly prototyped in PROLOG. Specialized property categories such as INVARIANT, INITIAL, CONSUMES, PRODUCES, and GOALS define the role of the predicates in transactions.

The TDL environment will contain essentially the same facilities as the SML environment. However, the emphasis of the prototyping tools is on (almost) full functional prototyping, attempting to derive PROLOG code directly from the transaction specifications using a library of standard PROLOG builtins.

2.3.3 Database and Transaction Development in DBPL

The third level of the DAIDA architecture is concerned with the efficient implementation of the conceptual design in a particular database programming environment. The database programming language DBPL [MRS84, ECKH85] is based on a data model supporting the predicative access and control of large shared data sets and has the system programming language MODULA-2 as its algorithmic kernel. DBPL data sets are made up of elements constructed by the data structures of MODULA-2, and can be restricted and queried by first-order expressions which can be abstracted as selectors resp. constructors [MRS84, JLS85]. DBPL data objects can be made persistent by declaring them in a database module which exports their type, value, and possibly transactions over them. Transactions serve as the unit of integrity, recovery, and concurrency control.

DBPL is distinguished from the upper-level languages in that (a) it does not support the inheritance mechanisms of generalization hierarchies, and (b) it has to realize predicative specifications as imperative programs. Therefore, we naturally have two major mapping tasks, structure and transaction mapping. TDL-to-DBPL mapping assistants can be based on theory developed in the implementation of semantic data models [WEDD87] and predicative programming [HEHN84]. The need for a separate database programming level arises since there are usually several alternatives how to realize the mapping tasks (a) and (b), and current theoretical understanding does not suffice to choose automatically among them.

Additionally, DBPL introduces the new structuring principle of modules for software; thus, there is a software organization problem to be solved at the DBPL level itself. Finally, the current version of

DBPL does not have a corresponding concept for scripts; one idea is to make TDL scripts directly executable in the DBPL environment, another idea is to map them to user manuals for DBPL.

Besides the mapping assistants and a GKBMS interface, the DBPL environment will contain structure-oriented editors, database design aids, and code management facilities.

2.3.4 Integrative Tools

Initial implementations of all three languages are avaiable or will be completed during the current project year. Integration among the three levels is provided by several components in the architecture. As mentioned, PROLOG-based prototyping facilities enable validation of the SML and TDL models. Mapping assistants support the user in transforming SML descriptions into TDL designs, and TDL designs into DBPL implementations. Finally, the Global KBMS provides a shared service intended to:

- coordinate development within and between the three language levels,
- assist in ensuring consistency concerning terminology, validation and formal correctness,
- support maintenance (i.e., corrections, adaptations, and enhancements).

This service consists of specialized documentation data structures, inferences mechanisms, database and dialog management facilities, and will be described in some more detail in section 3.

2.4 A DAIDA Development Example

The following example (figure 3) is based on an experiment [JR87] in which we applied the DAIDA methodology manually to information systems applications in project meeting organization. An SML *world model* starts from the activity, Meeting, within a project and describes its related activities and entities in a real world with time. Among other things, meeting preparation, conduction, and follow-up is different for people in different roles, namely organizers and other participants. Based on this observation, the SML *system model* is positioned in the world model in two functional parts (also called system activities or views), one supporting an organizer, the other a participant.

The combined world and system model are mapped to a TDL *design model*. The role of the system model within long-term world model activities is represented by a script, office-internal meeting schedule, certain aspects of other activities and data are mapped to data classes, transaction classes, and their corresponding constraints. Within the TDL model, data class hierarchies and corresponding transaction hierarchies must be synthesized from the mapping results, to achieve an integrated conceptual design; this could be called a particular strategy for *view integration*, to be supported by the TDL knowledge-based design assistant. In our example, we detected that from the various outputs of meeting we could compose a conceptual office document database, consisting of expense notes, working papers, invitation letters, minutes, and the like.

The integrated design model is then mapped to a DBPL *database structure and transaction design*. There are several decisions and trade-offs involved in mapping the above-mentioned generalization hierarchy of papers, and the corresponding hierarchy of transactions, to a set of relations, views, integrity constraints, and database transactions in DBPL (see section 3.1).

3. THE KNOWLEDGE BASE MANAGEMENT ENVIRONMENT

A Global Knowledge Base (GKB) represents the history of a particular database and its associated application software in an evolving real-world environment. This, we call a *software world*. Thus, the GKBMS is a management system to create and maintain knowledge about software worlds. Following this chain of reasoning, it appears natural to employ the language, SML, as an internal knowledge representation language for the GKBMS. In this section, we illustrate the interaction of knowledge-based components such as mapping assistants with the GKBMS, and describe a first partial implementation.

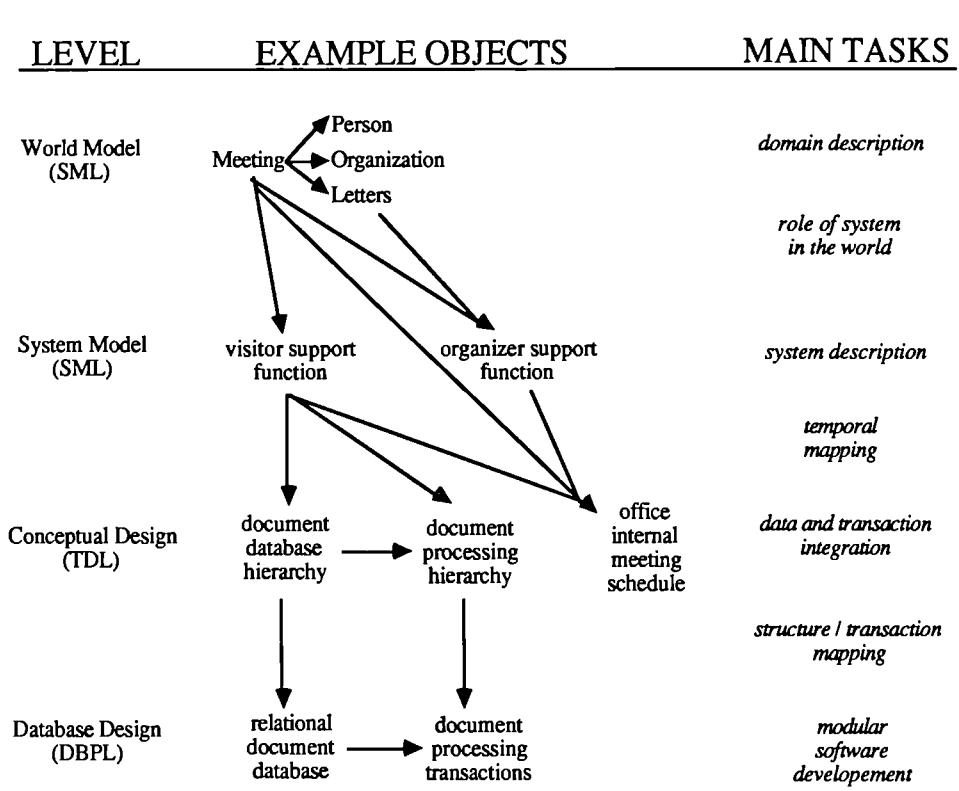


Fig. 3: Overview of an example development

3.1 Interaction of Mapping Assistants and Global KBMS: An Example

In this subsection, we use a small subexample from section 2.4 to demonstrate the intended integration of knowledge-based components in DAIDA. The example concerns the implementation of TDL entity class hierarchies as DBPL data structures. The entity classes are listed below and sketched in figure 4.

```
ENTITY CLASS Papers WITH
        date: Date
        author: Person
        content: Text
END { Papers };

ENTITY CLASS Invitations ISA Papers WITH
        receiver: Organization
        sender: Person
        forProject: Name
        meetDate: Date
    INVARIANT
        author = sender
END { Invitations };
```

To map faithfully the generalization hierarchy of entity classes, the DBPL programmer must create record structures, relation types and variables. There are several alternatives which would be shown to the user by a mapping assistant [WEDD87]. Here, we assume that just one relation, InvitationRel, is created from the hierarchy.

```
TYPE InvitationType = RECORD
                        paperkey: Surrogate;
                        invitationkey: Surrogate;
                        date: DateType;
                        sender: NameType;
                        content: Text;
                        receiver: OrganizationType;
                        forProject: NameType;
                        meetDate: DateType;
                      END;
    InvitationRelType = RELATION paperkey, invitationkey OF InvitationType;
```

The artificial keys (surrogates) are generated since TDL does not identify objects associatively; the programmer can replace these keys by associative ones. In a second step, constructors are introduced to build views that simulate those objects of the generalization hierarchy not physically represented in DBPL relations. The constructor ConsPapers computes a relation which projects InvitationRel on the paper attributes (note that the author attribute was left out in InvitationRel due to the constraint in the definition of the entity class Invitation).

```
CONSTRUCTOR ConsPapers FOR Inv: PaperType;
BEGIN
    < i.paperkey, i.date, i.sender, i.content> OF EACH i IN InvitationRel: TRUE
END;
```

Such a constructor would, for example, be used to implement elegantly a transaction that generates general papers (not invitations). Conversely, transactions for inserting into specialized relations must also consider the predecessor relations in the IsA hierarchy.

An idea of the user interface we are implementing is given in figure 4, mirroring a particular moment in mapping a TDL hierarchy to DBPL relations. Three windows show design objects in TDL and DBPL (the latter an incomplete code frame in which the user should insert the relation key), one a graphic representation of the TDL generalization hierarchy, and the last one an excerpt of a dependency graph relating all objects visible in windows on the screen. All of the graphical functions have already been realized in prototype versions but have been simplified in figure 4 for understandability.

As indicated by the figure, the interaction between mapping assistant and GKBMs involves three steps, each alternating between user control, tool (=mapping assistant) control, and GKBMS control:

(1) The mapping assistant sets a *focus* in the dependency graph of the GKB (here: the generalization hierarchy to be mapped) and requests information about the environment of design objects and dependencies (by ASK operations concerning TDL and DBPL levels).
(2) Given this environment, the mapping assistant, supported by the GKBMS, offers the user a *choice* of applicable mapping rules of which the user selects one by a TELL (here: to map the hierarchy to a single relation). This choice is not shown in the figure but would also be documented in the dependency graph .
(3) The mapping assistant executes the selected mapping rule by creating or changing objects (here: DBPL objects), TELLs the GKB about the changes, and returns control to the user for *refining* those parts of the new objects that could not be completed automatically, and TELLing the GKBabout the refinements.

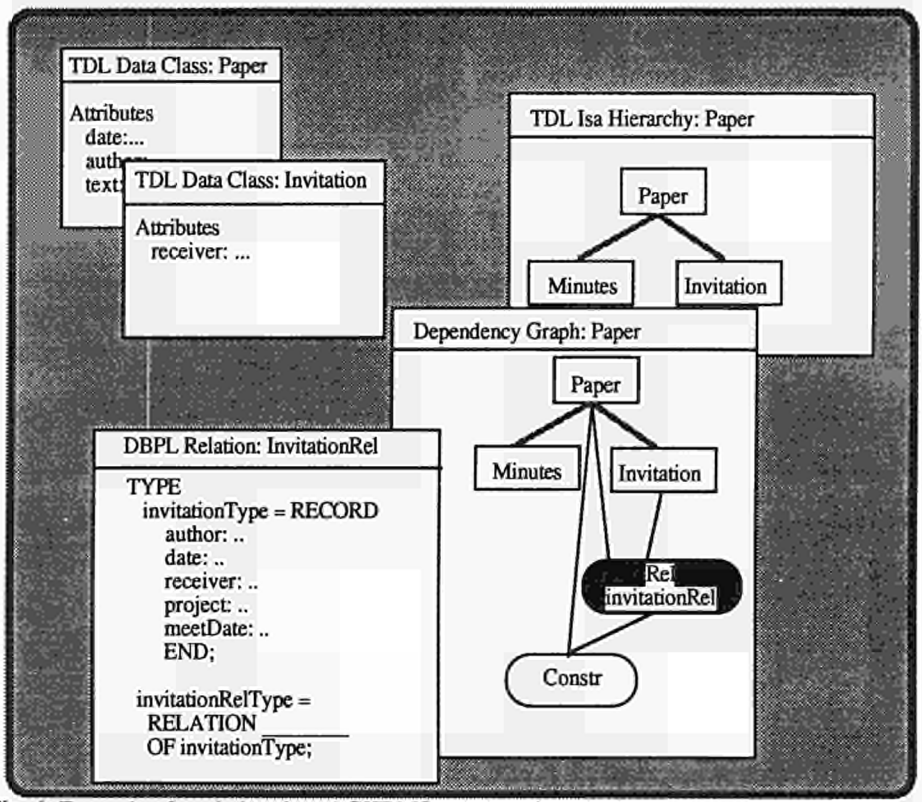


Fig. 4: Example of a window-based GKBMS usage environment

3.2 Functional Definition

As described in section 2.1, a KBMS can be functionally characterized by its ASK and TELL operations. The definition of TELL describes the knowledge structures the KB can be informed about; in SML, these are hierarchically organized object descriptions and assertions over time. The kind of knowledge inserted into the KB includes (formal definitions can be found in [JR87]):

- knowledge about SML, TDL, and DBPL design objects,
- knowledge about the design tools offered by the development and mapping assistants,
- knowledge about the history of development and maintenance decisions, represented as justifications for design objects.

A design justification represents a design decision and its rationale by relating a new (version of a) design object to the objects it was derived from, including the rule object (if more than one rule was applied, several justifications must be created); a time stamp represents the recording time for the justification to deal with version management issues. A set of justifications can be viewed as a *dependency graph* similar to those used in [SS77, DOYL79] which completely represents the design process. It is presently expected that cyclic dependency graphs will be forbidden. The graph can be partitioned into subgraphs by defining dependency classes similar to object classes.

The ASK operation on an SML knowledge base allows querying for properties of the objects or truth values of assertions. Specifically, the GKBMS ASK operation will allow:

- the retrieval of design object and design rule representations,
- the tracing of relationships among design decisions.

Given the above structure of the TELL operation and remembering the example in section 3.1, there are at least three abstraction levels a GKBMS user (human or program) could be interested in:

- *object level:* Here, the user wants to retrieve one or more design objects (or design rules) in the source-code syntax of their original language (i.e., SML, TDL, or DBPL). Object level operations work in the "software world" rather than in its GKBMS representation. If we have a partially specified object, this representation would be a *code frame*. The code frame representation is typically employed by the human user for programming, or by programs such as language-sensitive editors, compilers, or interpreters.
- *object representation level:* Here, the user wants to retrieve one or more SML representations of software world objects. This representation allows a more compact viewing of the design objects and is the one on which the pattern-matching part of design-rule assistants operates.
- *dependency graph level:* Here, each design object or design rule is viewed as an uninterpreted atomic or molecular node in the dependency graph that represents the design process. By molecular we mean that there could be several abstraction levels at which dependencies are defined, e.g., at the level of relation or at the level of attribute. The dependency graph representation plays a central role in maintenance support. It will be used to explain designs by tracing back their development history, and for propagation of change.

When the user (human or system) ASKs the KB, it is therefore very important to specify the level at which the answer is desired. A second important consideration is the size of the KB. Even looking at very small examples, there is a very large number of design objects and dependencies involved in designing a system through all levels. Besides choosing the correct level of abstraction, the user must also be able to define a working area or *focus* in which he/she/it is interested. Objects close to the focus (e.g., neighbors in the dependency graph representation) are expected to be more interesting to the user than those further away.

3.3 Architecture and Implementation Environment

The GKBMS will be designed and implemented in two layers: the GKBMS kernel and a distributed interface. The graphics interface must be separated from the query language, to make the query language and kernel portable between SUN-UNIX and VAX-VMS. Internally, the query language must be separated from the kernel since there are different kinds of users (human and machine). The *GKBMS kernel system* will provide:

- a typed *design object base* which contains the hierarchically organized object-representation level SML representation of design objects. It is currently foreseen that the object level representations (i.e., the software world itself) will be managed by each environment individually. This implies that each environment has to provide a *two-way translation and access mechanism,* so that access to design objects at other levels (or to previous and incomplete versions of own design objects) can be effected.
- a typed *design dependency base* which contains the dependency graph. The edges of this graph are typed to account for the different kinds of design decisions . This appears necessary to manage the complexity derived from the very large number of possible relationships an object is involved in.
- a *design rule base* which contains abstract representations of all the rules, methods, and tools contained in the individual mapping and design assistants. The design rule base provides a common specification language and storage for tools that could be implemented in very different fashions. The rule base specification also serves as a basis for determining when a particular tool is applicable, and as a basis for explanation facilities.
- A *query evaluation system* and an *integrity checking system* for the TELL and ASK operations. The query evaluation system will support the operations indicated in the examples but also object-oriented secondary storage access. The integrity checking subsystem will check typing and assertions of design objects; other consistency maintenance tasks will be included later in cooperation with the belief maintenance system.
- A justification-based *belief maintenance system* intended to document rule applications, ensure consistency of the knowledge base (and, therefore, hopefully of the designs represented in it) and to propagate changes by interpreting change propagation as guided search for culprits of integrity violations.

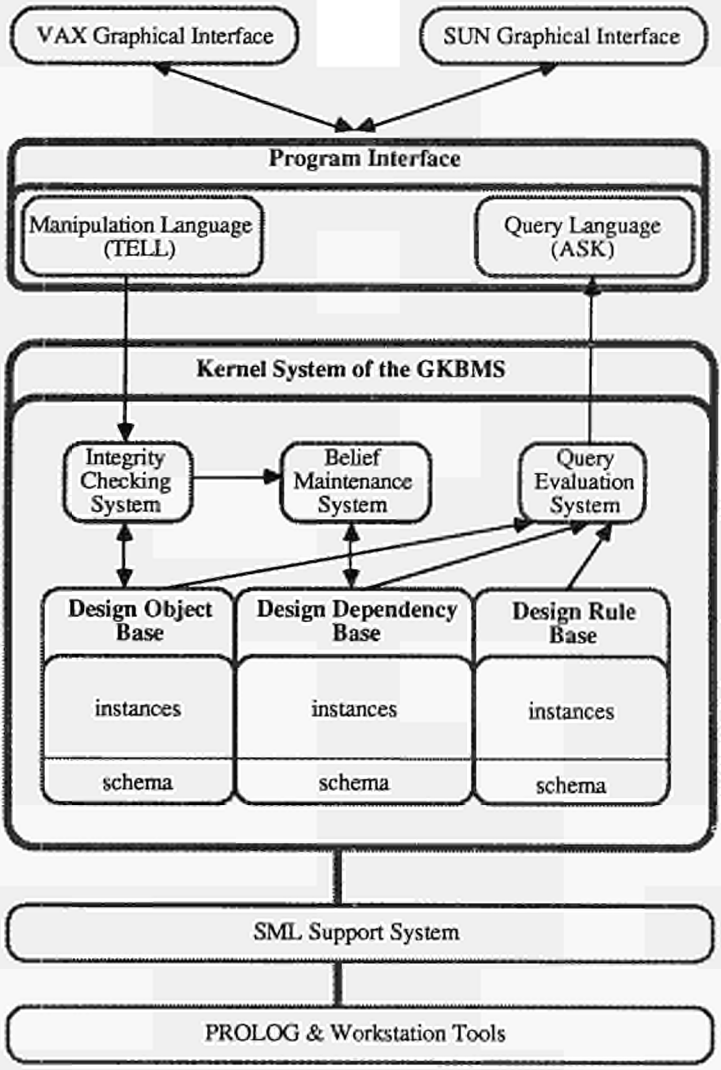


Fig. 5: Layered architecture of the global KBMS

The *program interface* will offer ASK and TELL operations as well as more complex composite operations in a syntax based on the SML assertion language. It will be translated to BIM-PROLOG, enhanced with object-oriented built-in predicates. At the dependency-graph level, the GKBMS query language will contain primitives for graph search, to aid in the tracing of dependencies and to facilitate the implementation of the two graphics interfaces. This architecture implies that mapping assistants must be specified (not necessarily implemented) in SML.

Finally, the *user interface system* will provide window-based support for text, (simple) graphics, and menu-based interaction of the GKBMS with human system developers (possibly also with end users who may have questions about the system). Besides formal query language and menu-based expression of queries and assertions, there will also be mouse-based selection and copying of

objects, as usual in a workstation environment. Interface objects will be specified in SML, and layout information can be stored with this specification. Dialog sequences will be managed following an XS-2-like approach [BHMN85].

The DAIDA workplan foresees two distinct prototypes, the second building on the capabilities of the first. The **first prototype**, whose implementation is currently underway, will assume an *empty rule set* ; consequently, design and mapping rules cannot be entered, checked for consistency, or retrieved from the first system version. Thus, the first prototype is approximately a design *database* with fairly sophisticated data structures (those of SML, including dependency graphs), query languages (allowing recursion), and user interfaces (allowing focusing and graph traversal). The documentation database will be filled manually, as designers document their decisions. Under these circumstances, maintenance support is limited to the (human) review of previous design processes that could be applied analogously. Nevertheless, these are very useful services, in particular because the GKBMS can be used by each individual environment to manage intermediate results.

The first prototype forms the kernel of the **second prototype** which will focus on rule processing, especially in the form of mapping assistant rule specification, design explanation and maintenance support. Through interchange with the mapping and design assistants, much of the documentation will be automated. Consistency checks and propagation of change through selective rule application replay will be offered by an extended belief maintenance system [DOYL79, DEKL86, DJ85, RICH84] with support for abstraction. The documentation of design rule applications for subsequent replay in the case of maintenance has also been proposed elsewhere but has met with limited success in a general environment where there is usually not enough knowledge to restrict the choices [BALZ85]; it is hoped that more knowledge can be brought to bear in tdatabase programming.

4 CONCLUSION

The use of automated transformation support in software development environments is a well-recognized concept but few practical implementations exist; one exception is the REFINE system developed at Kestrel institute [Smith et al. 1985], others are being developed in ESPRIT projects such as METEOR. Major features distinguishing DAIDA's approach include the emphasis on

- knowledge-based support rather than automation of analysis, design, and programming,
- exploitation of knowledge about a particular domain: data-intensive information systems,
- the temporal component of knowledge bases,
- to a large degree, adaptation of existing languages and tools with similar philosophies.

For the research partners, this context has presented a large number of unique and challenging research questions, including those of how to transfer theory to practice. For the software houses involved, DAIDA appears as an attractive approach to evaluate and extend their logic programming, object-oriented and graphics tools, and to upgrade their commercial software development environments by practical and extensible knowledge base management components.

REFERENCES

[ALLE83] Allen, J.F. (1983). Maintaining knowledge about temporal intervals, *Commu. ACM 26*, 11, 832-843.

[BALZ85] Balzer, R. (1985). A 15 year perspective on automatic programming, *IEEE Transactions on Software Engineering SE-11*, 11, 1257-1267.

[BHMN85] Biagioni, E.S., Hinrichs, K., Muller, C., Nievergelt, J. (1985). Interactive deductive data management - the smart data interaction package, in Brauer, W., Radig, B. (eds.): *GI-Kongreß Wissensbasierte Systeme*, Springer, 208-220.

[BL86] Brachman, R., Levesque, H. (1986). Knowledge level interfaces to information systems, in Brodie, M.L., Mylopoulos, J. (eds.): *On Knowledge Base Management Systems*, Springer, 13-34.

[BMMS87] Borgida, A., Meirlaen, E., Mylopoulos, J., Schmidt, J.W. (1987). First Version of TDL Design, Esprit Project 892 (DAIDA), Cretan Computer Institute, Iraklion, Greece.

[BORG87] Borgida, A., Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y. (1987). The software development environment as a knowledge base management system, appears in Schmidt, J.W., Thanos, C. (eds.): *Foundations of Knowledge Base Management,* Springer-Verlag.

[DEKL86] de Kleer, J. (1986). An assumption-based TMS, *Artificial Intelligence 28,* 2, 127-163.

[DJ85] Dhar, V., Jarke, M. (1985). Dependency-directed reasoning and learning in large systems maintenance, *IEEE Transactions on Software Engineering,* to appear.

[DOYL79] Doyle, J. (1979). A truth maintenance system, AI Memo 521, MIT, Cambridge, Mass.

[ECKH85] Eckhardt, H., Edelmann, J., Koch, J., Mall, M., Schmidt, J.W. (1985). Draft Report on the Database Programming Language DBPL, Johann Wolfgang Goethe-Universität, Frankfurt, FRG.

[GBM86] Greenspan, S., Borgida, A., Mylopoulos, J. (1986). A requirements modelling language and its logic, in Brodie, M.L., Mylopoulos, J. (eds.): *On Knowledge Base Management Systems,* New York: Springer-Verlag, 471-502.

[HEHN84] Hehner, E. (1984). Predicative programming, *Comm. ACM 27,* 2, 134-150.

[JARK86] Jarke, M., ed. (1986). DAIDA Global Design Report, Esprit Project 892 (DAIDA), Johann Wolfgang Goethe-Universität, Frankfurt, FRG.

[JLS85] Jarke, M., Linnemann, V., Schmidt, J.W. (1985). Data constructors: on the integration of rules and relations, *Proc. 11th Intl. Conf. Very Large Data Bases,* Stockholm, 227-240.

[JR87] Jarke, M., Rose, T. (1987). Global KBMS Design and Development Strategy, Esprit Project 892 (DAIDA), Universität Passau, FRG.

[JV84] Jarke, M., Vassiliou, Y. (1984). Coupling expert systems with database management systems, in Reitman, W. (ed.): *Artificial Intelligence Applications for Business,* Ablex, 65-85.

[MRS84] Mall, M., Reimer, M., Schmidt, J.W. (1984). Data selection, access control, and sharing in a relational scenario, in Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): *On Conceptual Modelling,* New York: Springer-Verlag, 411-436.

[MBW80] Mylopoulos, J., Bernstein, P.A., Wong, H.K.T. (1980). A language facility for designing interactive data-intensive applications, *ACM Trans. Database Systems 5,* 2, 185-207.

[RI85] Rauch, E., Insel, B. (1985). PRADOS -- die SCS Software-Engineering-Umgebung, in Balzert, H. (ed.): *Moderne Software-Entwicklungssysteme und -werkzeuge,* BI Wissenschaftsverlag, Mannheim, 253-262.

[RICH84] Rich, C. (1984). A formal representation of plans in the Programmer's Apprentice, in Brodie, M., Mylopoulos, J., Schmidt, J.W. (eds.): *On Conceptual Modelling,* Springer, 239-269.

[RS77] Ross, D.T., Shoman, K.E. (1977). Structured analysis for requirements definition, *IEEE Transactions on Software Engineering SE-3,* 1.

[SKW85] Smith, D.R., Kotik, G.B., Westfold, S.J. (1985). Research on knowledge-based software engineering environments, *IEEE Trans. Software Engineering SE-11,* 11, 1278-1295.

[SNOD86] Snodgrass, R. (1986). Research concerning time in databases: project summaries, *SIGMOD Record 15.*

[SS77] Stallman, R.M., Sussman, G.J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, *Artificial Intelligence 9,* 2, 135-196.

[WEDD87] Weddell, G. (1987). Physical Design and Query Compilation for a Semantic Data Model, Ph.D. Thesis, University of Toronto, Canada.

Project No. 300

THE REQUEST DATABASE FOR SOFTWARE RELIABILITY AND SOFTWARE DEVELOPMENT DATA

Chris DALE

National Centre of Systems Reliability, United Kingdom Atomic Energy Authority, Wigshaw Lane, Culcheth, Warrington, WA3 4NE, United Kingdom

REQUEST (REliability and QUality of European Software Technology) is carrying out a variety of research which has a common requirement for data from the industrial development and use of software. A data collection and storage activity has been established by REQUEST to ensure that the data required is collected, stored and made available for analysis, but also to experiment with the direct role of such a database in improving European software quality. This paper describes the current status of this activity.

The first stage of the database work has involved identifying the metrics and other software engineering data which it is desirable to collect; developing a data model incorporating these items; and from this devising methods of collecting and storing the data. Much of this work is now complete, so that data is being collected from a number of industrial data providers for storage in the REQUEST database machine at Winfrith.

The second stage will involve developing and using data analysis methods which will produce useful feedback to data providers and other users of the services, as well as providing analysis facilities to enable the research work of REQUEST on software quality and reliability to continue. Most of this development work is now underway.

It is intended that services based upon the REQUEST database work will be available at a future date, to assist in future enhancement of European software quality.

1. THE REQUEST PROJECT

The REQUEST project is a collaboration of seven companies from five EEC countries, aimed at providing improved and validated techniques for measuring and modelling software quality and reliability, supported by appropriate prototype tools. The project is primarily concerned with developing metrics and models which span as much of the development life-cycle as possible from

specification to life-use, and with providing useful and timely information for project quality management decision making and control.

In particular, REQUEST aims to develop:

- a constructive quality model (COQUAMO) which can be used to predict and control quality characteristics throughout the software development process;

- improved reliability models which incorporate testing information into models;

- new approaches to modelling the reliability of single version and multiple version software systems, which incorporate development process and software structure information by the use of "explanatory variables" (ie variables which are used to adjust a basic reliability model);

- standards and guidelines for software data collection and analysis;

- a database of software data which is used to validate proposed metrics and models.

It is an objective of the project to produce prototype tools, where appropriate, to enable ready use of metrics and models.

In pursuit of these aims, the project has been organised as three sub-projects:

1 Quality measurement, modelling and prediction;

2 Reliability measurement, modelling and prediction;

3 Data collection and storage.

This paper concentrates on the third of these sub-projects, describing the current status of this activity. It is important to appreciate however, that the data collection and storage activity was initiated principally to satisfy the needs of the researchers working in the other two areas. It has become increasingly clear that the work of sub-project 3 is of great potential value to researchers and others outside the REQUEST project, with a wider range of applicability than to the relatively narrow technical interests of REQUEST researchers. There is, therefore, a strong exploitation opportunity associated with this data collection and storage part of the REQUEST project.

2. THE DATA COLLECTION AND STORAGE SUB-PROJECT

The purpose of any data collection activity, in software engineering as elsewhere, is to enhance understanding of the phenomena about which data are collected. Thus the reason REQUEST wishes to collect data is to develop an increased understanding of software quality and reliability, based upon data collected over a period of time from a large number of projects. This same

data, however, has other uses which will be of more obvious and immediate benefit to those organisations providing the data. In this way, collection and elementary analysis of data from a project can significantly enhance management understanding and control of the problems of that particular project. It is these short-term benefits which persuade data providers to allow REQUEST to use their data, rather than the longer-term, more esoteric benefits sought by REQUEST researchers.

The prime objective of the REQUEST data collection and storage activity is to provide a data resource to support the needs of other REQUEST sub-projects. This necessitates:

- identification of the data requirements of REQUEST researchers;
- identification of sources of the required data;
- establishment of procedures for collection of the data;
- establishment of a database system to store the data and permit analysis by REQUEST researchers.

When work began on identifying the data requirements, it was quickly realised that a very general, highly flexible data model would be needed, because

- the research will inevitably generate new data requirements as it progresses
- the researchers require industrial data, so that the data model needs to cope with data from a wide range of industrial software development practices.

The development of this model proved to be a very large and complex task, but the model now forms the basis of both the design of the REQUEST database, and the data collection manual which has been developed to facilitate collection of the data. The data collection manual is one example of the beneficial co-operation there has been between this project and the Software Data Library Project (SWDL) funded under the UK Alvey programme of research and development in advanced information technology.

Identification of sources of data is one of the aspects of the work which is carried out by the REQUEST data collection team. This team has members in the UK, France, Denmark and West Germany who are responsible for co-ordinating REQUEST data collection within their respective countries. Their duties include the initial identification of potential data providers, holding meetings and giving presentations to encourage the provision of data to the project, together with certain activities associated with the data collection itself. In the UK, data collection team activities are carried out by the Software Data Library project on behalf of REQUEST.

Data collection procedures are based around the data collection manual, which describes the data to be collected and how the forms contained in the manual should be used. An important aspect of data collection from a diverse population of data providers is the notion of comparability data. In order to enable comparison and analysis of data from differing projects, it is essential

to have a clear understanding of the definitions used in each individual project from which data is collected. A useful example of this idea is the **lines of code** metric, which can be defined in many different ways. Unless it is known how it is defined on each individual project, it is impossible to make intelligent comparisons between projects. This same principle applies to all data which are collected for the REQUEST database.

Having determined the data requirements, and the more general requirements of the REQUEST researchers, it has been possible to develop the (hardware, software and human) system which comprises the REQUEST database. As with other aspects of the work, this has been determined primarily by the needs of REQUEST researchers. There is, however, another important group of people who need to be satisfied: the data providers. It is necessary to provide services to those who provide data to the project, in order to encourage them to do so, and to repay them when they have provided data. The needs of such users have also been taken into account in defining the system.

The REQUEST data collection and storage sub-project can thus be summarised as a database which collects data from data providers and provides services to REQUEST researchers and to data providers.

3. THE REQUEST DATA MODEL

The software data model developed by REQUEST (in association with the SWDL project) provides a basis to the database design and to data collection which is as comprehensive and flexible as possible at the current time. The model consists of

- a definition of each entity for which data is collected (eg **component**, **task**)
- a definition of the relationships between the entities (eg a **component** is modified by an **operation** which may be due to a **software engineering note**)
- a list of attributes associated with each entity (eg a **component** may have the number of **lines of code** counted, a **task** may have the **development method** used recorded).

Figure 1 provides a top-level generic view of the data model.

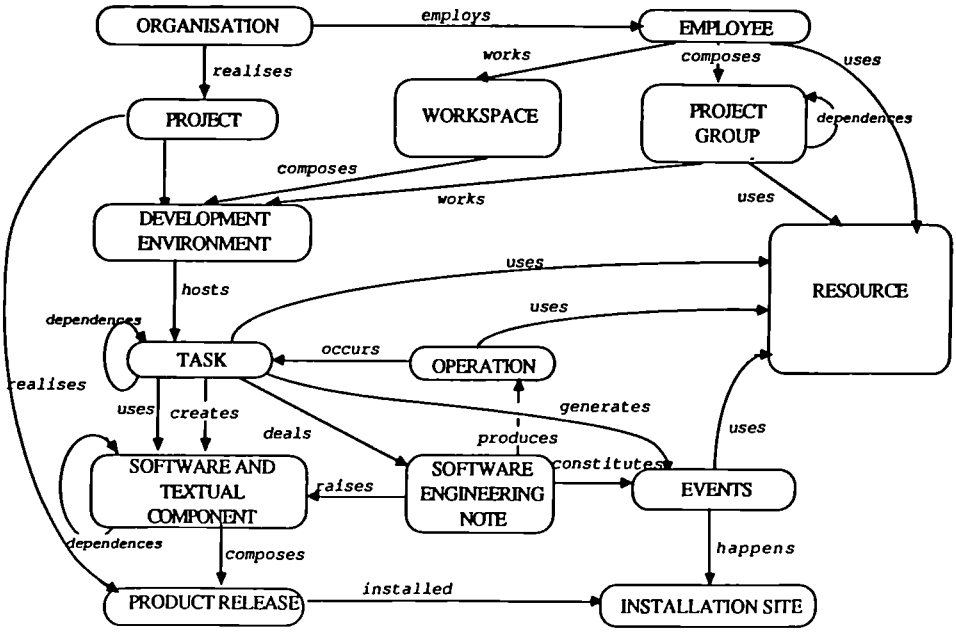


FIGURE 1
REQUEST generic data model

Some of the boxes in Figure 1 represent entities, whilst others represent groups of entities, as explained below.

Organisation	-	a body which controls projects.
Project	-	an activity which develops product releases.
Employee	-	a person employed by an organisation.
Project Group	-	a group of employees.
Installation Site	-	the site at which a particular product release is being used.
Product Release	-	a collection of software and textual components delivered to customers.
Task	-	a planned software engineering process (eg design of a software component).
Operation	-	an unplanned software engineering process (eg dealing with an incident).

Development Environment	-	the environment in which software engineering tasks are carried out.
Workspace	-	the physical environment in which an employee works.
Resource	-	this comprises four entities recording usage of consumable resources (eg man hours) and non-consumable resources (eg men) by employees and project groups.
Software and Textual Component	-	this comprises three entities recording data about items created by any of the software engineering processes involved in producing a product.
Software Engineering Note	-	this comprises three entities recording data about incident reports, fault diagnoses or change requests.
Events	-	this comprises three entities recording data about groups of incidents, faults or changes.

Having outlined the entities of the model, attention now turns to the attributes of these entities which are recorded. These are shown in Figure 2, together with the entities to which they apply.

The attributes (shown without underlining) appear in two ways in Figure 2. Attributes within parentheses actually refer to groups of individual metrics, whereas attributes without parentheses are themselves metrics. For example, the attributes of **component** include (Quality) and Complexity. Here, (Quality) represents the individual metrics Reliability, Maintainability, Reusability, Extendability, Efficiency, Usability, Integrity and Generality - each of which is measured on a subjective 5-point scale - whilst Complexity is itself a metric, also measured on a 5-point scale.

The model which has been outlined above is a very general one and it is thus necessary to customise the model to the practice of any particular data provider, in order to make the task of data collection a practicable one. This customisation process can be thought of as a "pruning" of the "tree" shown in Figure 2. Because there is a strong correspondence between the data model and the data collection forms, it is a relatively straightforward task to customise the forms to the particular data collection activity by simply removing those forms or parts of forms which are not needed.

4. THE REQUEST DATABASE SYSTEM

The REQUEST database runs on a DEC MicroVax at the United Kingdom Atomic Energy Authority's Winfrith site in the South of England, using the INGRES DBMS. All data collected will be resident on this machine, but for reasons of confidentiality and security all identifiers of people, programs, projects or organisations will be removed and replaced by codes before entry to the system.

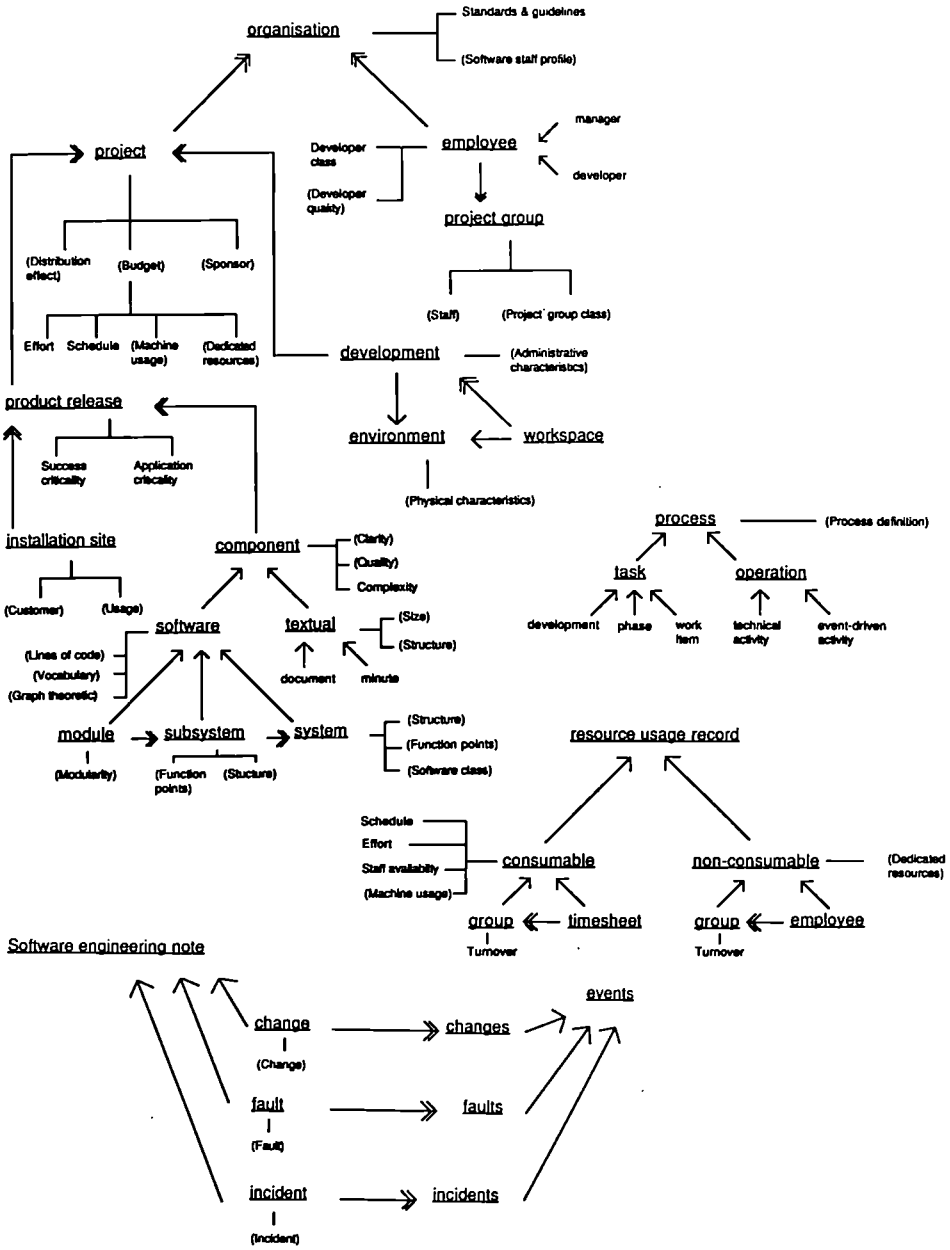


FIGURE 2
Entities and attributes of the REQUEST data model

Direct access to this system will be limited to named researchers working on the REQUEST project.

Users of the system who are not REQUEST researchers will gain access to data by connecting to a separate machine at Winfrith, which will (by prior arrangement) contain data to which they have access authority, or an empty copy of the database to allow entry of new data to a holding database. This strategy has been adopted to afford maximum protection to any commercially sensitive information stored on the system.

The basic REQUEST database system is now operational, based upon the INGRES facilities and commercially available statistics packages. Work to be carried out over the next few months will lead to enhanced MMI and analysis facilities.

5. USER CONTACT WITH THE REQUEST DATABASE SYSTEM

There will be many kinds of users of the system, but they can be divided into three distinct categories - REQUEST researchers, data providers and others.

The researchers involved in the REQUEST project will clearly receive the best level of service, since the system has been designed with their needs in mind. They will be the only ones with direct access to the MicroVax, but even they will not be told to which particular people, programs, projects or organisations the data they are accessing refers.

Data providers are another group who will have a wide range of services available, in recognition of the contribution they have made to the REQUEST project. One of the chief benefits to data providers is the verification and analysis report - for every data set submitted, a report will be produced for the data provider giving an analysis of the data submitted, aimed at both providing useful feedback to the data provider and checking that there are no unexplained anomalies in the data.

The verification and analysis report consists of a series of statistical analyses of the provider's data, examined both in isolation and in comparison with other relevant data in the database. Once the initial analyses have been performed the report is reviewed by an experienced statistician and a software expert, who will add to the report their comments, explanations and queries on particular features of the data, such as values detected as anomalous.

Examination of and agreement with the content of this report by the data provider is a vital part of the data verification process employed by REQUEST. The report will also be valuable to the manager of the project to which the data relate, as it will help him to identify, understand and diagnose particular problems which have occurred in his project, and take appropriate remedial action. It will be useful to the organisation providing the data in a more general sense by helping to identify problems with the way in which they develop software, and enabling them to compare their performance with that of their competitors.

Other services for data providers include:

1 Data providers will be able to view on-line the data they have submitted.

2 Their data will be stored at least until the end of the REQUEST project.

3 As part of their contact with REQUEST, they will automatically receive a great deal of technical advice on software engineering data collection.

4 Periodically REQUEST will publish summaries of database content.

5 Statistical queries can be submitted to the database manager for processing. This service cannot currently be provided on-line for reasons of confidentiality and security, because these queries would relate to the entire database. Such requests for analyses will be subject to the constraints imposed by confidentiality and available resource.

6 Data providers will be able to carry out on-line analyses of the data they have submitted.

7 Data providers will be able to carry out comparisons of their data with summaries of the database on-line.

8 At a later date it is hoped to provide access to "typical" project data; considerations of confidentiality prevent this at the current time.

9 On-line data entry will be available to those who find this convenient.

The "others" category of user will have more limited facilities - in many cases, none at all. Currently it is proposed that groups of bona fide researchers approved by the REQUEST project will be able to obtain data for analysis from the REQUEST database, subject to conditions such as obtaining the permission of the data providers affected. These facilities will clearly be limited by the resources available within the REQUEST project, though in many cases it may be possible to establish a quid pro quo, so that there is some benefit to REQUEST in return for services provided - the most obvious example of this is where data is provided, as discussed above.

It is hoped that the potential for providing a service based on the database can be fulfilled in the future, but the current priorities are to keep two groups of people happy - the REQUEST researchers, and the data providers.

So far only user services have been discussed in this section. Another important contact between REQUEST and users is the provision of data to REQUEST. Here the interface is normally a human one - a member of the REQUEST data collection team representing REQUEST, and a "data collection supervisor" representing a data provider. The precise nature of this relationship varies a great deal, because of national differences and because of differences between

data providers. The following provides a general framework for the relationship, which is tailored to individual requirements:

1 Identification of data source - a telephone call, letter, or other personal communication initiates the contact by identifying a potential data provider.

2 Initial contact presentation - a meeting is held to explain the REQUEST data collection activity and provide enough information for the organisation to decide whether to become a data provider.

3 Agreement visit presentation - the agreement to provide data is formalised, the data model is customised, and the detailed arrangements for collection of data are established.

4 Data collection visits - batches of data are collected at agreed intervals, with some initial checking and analysis.

5 Verification report approval - the data provider keeps one copy of the report, and signs a second one to confirm the veracity of the data, at the same time responding to any questions raised during the verification and analysis process.

An important aspect of this sequence of events is the amount of contact between the REQUEST data collection team member and the data collection supervisor, providing opportunities for the data provider to learn from the experience and expertise developed on data collection issues within the REQUEST project.

6 CONCLUSIONS AND FUTURE WORK

Among the major achievements of the REQUEST data collection and storage activity to date are the following:

- Development of a general data model of the software engineering process, incorporating the ability to compare data collected from projects carried out in diverse software engineering environments.
- Development of a database system based on this model, as a means of storing the data and providing the necessary services.
- Development of a data collection methodology, covering all aspects from initial contact with a potential data provider, to the verification of data using statistical and software engineering expertise.
- Enrolment of data providers in several European countries.

Achievements that we aim to be able to report during 1988, or before, include the following:

- Development of full capability to produce verification and analysis reports.
- Provision of analysis facilities to REQUEST researchers.
- Completion of data collection and verification cycle for a significant number of projects.
- Publication of reports summarising database contents.
- Establishment of a database service.
- Automation of certain aspects of the work.

The exploitation opportunities of the work that has been done and is planned are considerable, and extend beyond the scope of the reliability and quality interests of REQUEST researchers. There is a strong possibility of developing a generally available information service based upon the REQUEST database, which could be useful in many areas of software engineering. One example of this is the role that such a database could play in the area of certification, by assisting to identify the basis on which particular software should be certified.

Subject to the reservation that much of what has been developed has yet to be tested in "the real world", I feel that the achievements of this activity are considerable, and hope and believe that many other projects will be able to benefit at a future date from the work that has been done. I believe that the REQUEST database will prove to be an enabler of a great deal of valuable work in the area of software metrics and more generaliy, and hope to find a way of making appropriate services available to a wide audience at an early date.

ACKNOWLEDGEMENTS

I should like to thank all those in REQUEST sub-project 3, the REQUEST data collection team, and the SWDL project for their contributions to the work reported in this paper. There are many others, both inside the REQUEST project and outside, who have not been directly involved in the work but who have provided support, encouragement, advice, and (mostly) constructive criticism, whom I also thank.

Project No. 1609

SMART : A SYSTEM DESIGNER APPROACH TO EVALUATE THE PERFORMANCE OF COMPLEX FAULT-TOLERANT SYSTEMS

A.KUNTZMANN
CISI INGENIERIE
3, rue LECORBUSIER 94578 RUNGIS CEDEX
FRANCE

J.FIGUEIRAS
UNIVERSITY of CATALUNYA
BARCELONA
SPAIN

ABSTRACT

A Markov model considering physical and design faults is the basic support for the work presented . The model can be split into two sub-models dealing with physical and design faults separately .
The generalization of the evaluation approach to reliability-oriented systems is considered .
From this modeling approach, the development of a suitable tool for system designer is presented, devoted to the evaluation of a design architecture in terms of Reliability, Availability, Maintenability,Cost, Management resources .
This work is partially supported by the Commission of the European Communities under the SMART project .

INTRODUCTION

One of the most critical problems that is faced by fault-tolerant system production is how to monitor architecture design together with development process in order to meet performance criteria, ensuring cost effectiveness . Evaluation models encompassing physical faults and design faults introduced during the development process could help to solve this problem . The need for such an evaluation has recently be pointed out (1) .
Due to the complexity of current software and hardware designs, it is no longer possible to cope with design faults for critical applications using only a fault-avoidance approach, and fault-tolerance techniques for design faults should be considered in many cases .
Design diversity (2) is a possible approach that provides potential effectiveness for design fault-tolerance .
Some work has been done towards combined modeling of physical and design faults (1), (4), but, at the best of our knowledge, no evaluation of a system incorporating both types of faults and including the performance aspects has been carried out .
SMART , through data collection measurement analysis and modeling, will provide the system manager with techniques and tools for evaluation, prediction and optimization of applications that have to match constraining fault-tolerant objectives .

CURRENT STATE OF PRACTICE

In general, correctness of software and software reliability are correlated . Intuitively, this means that the fewer faults there are (left) in a piece of software, the less likely it is to fail . However, high reliability can be obtained in spite of lower correctness if the faults are of a kind that seldom manifest themselves as failures, or if the software has been built to check its results and repair the effects of the faults before failures occur .

One of the two main techniques for producing fault-tolerant software is the recovery block technique . At strategic points, if a problem is detected, the former state is reestablished, and the computation retried with backup software . The granularity of the prime and backup models varies with the application . At the University of Newcastle upon Tyne, a research project has been conducted to explore this technique (3) .
The N-version approach is the other important strategy to fault-tolerant computing . It uses several (in the simplest case, three independently developed versions of software that perform the same functions according to the same specifications . (2) describes a project where a specially instrumented environment has been set up in order to explore this approach .

Fault-tolerant approaches to software construction represent a costly investment in reliability, which is usually only justified when the cost of improving reliability by marginally increasing correctness would be even greater . The predicted reliability as a function of both input metrics expressing the marginal cost of improving correctness, and the cost effectiveness of fault-tolerant techniques are therefore of interest for applications where very high reliability is needed .
Performance engineering as a whole is becoming a very important and sensitive discipline for system managers and designers . Obviously, there is a lack of an integrated approach by which to judge the overall performance of systems, one of the reasons being the difficulty of access to meaningful data for analyze .
SMART initiative, an ESPRIT project supported by the CEC, is a trial to encourage the bridge between software science and fault-tolerant architecture evaluation . One of the first objectives is to analyze the feasibility of deriving techniques from already existing theories dealing with hardware configurations and adapting existing tools to take into account software characteristics . The METFAC tool (6) developed by one of the SMART partner will be the basis of work .
METFAC will have a twofold role : first, it will give evaluation support to assist the integration of component models into system level models by aggregation and successive refinement ; secondly, it will serve as starting point for achieving an efficient and usable tool for system designers .

EXPERIMENT

Let us consider a system composed of two independently designed computation lanes sharing a given set of input and producing separate output that are compared by a totally self-checking (TSC) monitor M . The output of the system is taken from lane 1 and is considered valid as long as no failure indication is given by M.
The faults under consideration are classified as follows :

- physical faults
  - in lanes
    - related
    - unrelated
  - in monitor
    - benign
    - latent
- design faults (in lanes)
  - unrelated
  - related

Design faults are viewed as domains in the system input space (sequences of input vectors) . In the fault model considered, domains in different versions are either disjoint (unrelated design faults) or coincident (related design faults) .

The behavioral model is obtained by combining the fault model with the maintenance strategy . An unsafe failure occurs if an erroneous output is given without failure indication (identical errors in both lanes or latent monitor fault). When a failure indication is issued the system stops for diagnosis . If a permanent fault is found, a maintenance operation starts .
In order to quantify the dependability of the system, we will use as a measure the unsafety **US(t)**, defined as the probability of having an unsafe failure over the first **t** time units of operation (ignoring the time spent in the safe down mode) .

THE SMART APPROACH

The SMART project has identified three major area to manage the development of the system designer tool :

- the characterization of fault-tolerant architectures against metrics defined towards three reference systems (product characteristics, management environment, and development process) in order to quantify software architectures against metrics . Each architecture will be considered as an aggregation of software/hardware components .

- the modelling of performance :
  . for a single component within the three reference systems,
  . in a framework combining the three reference systems,
  . for the whole system using a structural approach .

- the development of a performance monitoring tool focusing on :
  . a user friendly graphical interface,
  . the possibility for system designers both to express the constraints of the system and to build the preliminary architecture,
  . a research and mathematical interface for model builders,
  . a facility to analyze the attibutes of each components and the relationships between components,
  . a support to validate the prediction achieved .

This tool will be the framework to compose the results from various models for single components within each reference system and to perform both the aggregation of components based on the description of different fault-tolerant structures and the combination of models according to production rules .

Some fault-tolerant basic mechanisms considered as primitives will be available in the data base of the tool ; the designer will be able to build any system architecture from these primitives, to run the model and to compare the results (Reliability, Maintenability, Safety) with the requirements to be met .In a further step, some amendments should be suggested by the tool to improve one or two of the selected criteria .

Different modes of use may be identified along the different phases of the system life cycle

- the system designer identifies the requirements to be performed by the final product,
- the system designer defines a possible architecture for the system from basic fault-tolerant components,
- before the end of the design phase, a complete analysis of the proposed solution is achieved towards the different relevant metrics by either running the existing models or estimating the results from the expertise base and statistical procedures support ,
- going further through the life cycle, some data might be measured on the system to validate the approach and obtain a better set of fitted models .

Iterations are possible in each above mode .

METHODS

Methods to match the above defined objectives are split into four categories :

- methods to define a Metrics reference system,
- methods to characterize Fault-tolerant Architectures,
- methods to model system performance,
- methods to build an efficient and usable tool .

The Metrics Reference System

Software product metrics are compiled from existing sources and experiments .

Software management metrics are derived from the abundant descriptions available in the literature and economical studies .

Development process metrics, especially those describing fault-tolerant approaches, such as for example proof of correctness, validation coverage estimator, structural complexity at different stages of the life cycle , are developed in SMART project .

The reference system addresses the quantification of the interrelated areas :

- the properties of the software product being developed, including performance, reliability, availability, maintenability and reusability .

- the properties of the software management environment, focusing on constraints such as cost, calendar time, manpower and risks .

- the properties of the software development process, including the design methodologies used and the tools available . The need of validated software components relying on life cycle activities and the earliest condidtions of the development are carefully analyzed .

The characterization of Fault-tolerant Architectures

The general purpose is to give evidence through several examples that the selected set of metrics does characterize the design and development of fault-tolerant systems .

The following fault-tolerant techniques are considered :

- cold/hot redundancy,
- fault recovery and fault masking (by HW or SW),
- recovery block,
- new version programming,
- diversity .

Modelling and quantifying performance

The objective is to formalize performance as a composite result of parameters in a multi-dimension system .
A global modelling approach is built step by step .

The SMART tool

The tool is the framework to compose the results from various models for single components within each reference system . It is flexible enough to be modified and extended according to the expertise gained through various experiences .
The integration of such a tool into the development environment (PCTE) and the link with all the other tools used to develop the system is strongly taken into account to improve the efficiency .
The kernel of the SMART tool will be an improved version of METFAC taking into account the software behaviour through new production rools .

CONCLUSION

Pointing out that the major challenges for fault-tolerance systems are :

- explosive growth of complexity that will avoid rough duplication for economic reasons and make unefficient any unstructured testing approach,
- design faults avoidance,
- specification faults avoidance,

SMART tends to provide a complete metrication reference system by improvement and adaptation of the already existing results and an integrated tools set for monitoring fault-tolerant systems development : estimation, evaluation and prediction .

This global approach is based both on the extension of software reliability theory and on the improvements of existing techniques for performance evaluation of fault-tolerant architectures .

REFERENCES

(1) **J.C LAPRIE,** Dependability Evaluation of Software Systems in Operation,
IEEE Trans. Software Eng. vol SE-10 Nov 1984

(2) **A.AVIZIENIS,** The N-version Approach to fault-tolerant Software,
IEEE Trans. Software Eng. vol SE-11 Dec 1985

(3) **T.ANDERSON, P.A BARRETT, D.N HALLIWELL,** An evaluation of software fault-tolerance in a practical system,
IEEE Trans. Software Eng. vol SE-11 Dec 1985

(4) **A.COSTES, C.LANDRAULT, J.C LAPRIE,** Reliability and Availability Models for Maintained Systems Featuring Hardware and Design Faults,
IEEE Trans . Computers vol C-27

(5) **J.A CARRASCO,J.FIGUERAS, A.KUNTZMANN,** Evaluation of safety-Oriented Two-Version Architectures
Report under publication Jan 1987

(6)**J.A CARRASCO,J.FIGUERAS,** METFAC : Design and implementation of a software tool for modeling and evaluation of complex fault-tolerant computing systems
FTCS 16 Vienna July 1986

(7) **R.A SAHNER, K.S TRIVEDI,** A Hierarchical Combinatorial-Markov Method for solving Complex Reliability Models,
ACM/IEEE Fall Joint Computer Conf. Dallas Texas Nov 1986

(8) **V.KINI, D.P SIEWIOREK,** Automatic Generation of Symbolic Reliability Functions for Processor-Memory Switch Structures
IEEE Trans. on Computers vol CE-31 August 1982

(9) **SMART consortium,** Technical Annex of the contract January 1987

Project No. 938

IMPISH: a RDBMS extended to handle logical rules and documents.

Michel BOSCO, Michel GIBELLI
DIG-TL
CETE Méditerranée
B.P.39 13762 LES MILLES CEDEX, FRANCE

Abstract:

IMPISH is a software able to manage information which has been modelised in several ways. It will be used first in the Software Project Management Field.

The Codd relationship is the basic concept of the model, but some extensions of the usual relational languages have been designed and coded, in order to handle logical rules and documents.

It provides the user with several interfaces for traditional applications as well as expert systems.

1. INTRODUCTION

Those Information Systems based on a relational vision, or even 'EAR', of the real world, are, for the most part, implemented on relational DBMS.

However, the need to manage complex information and factual knowledge leads us progressively towards deductive data bases, and there is a great deal of work being done in this area. Some articles recently published show convincing results from projects undertaken in the field of the integration of relational systems and inference engines (1) (2) (3).

On the other hand, certain applications depend on the simultaneous management of modelised data (as in the case of the 't-uples' or the logical rules described in PROLOG, for example) and information which we describe as 'physical', such as documents and images.

The computer-assisted management of software projects is a typical application: the creation of a prototype management workbench (IMP Workbench, which stands for Integrated Management Process Workbench) (4) demonstrates the need for a host structure for an information system capable of managing modelised data and texts.

In this article we will present a first prototype of this structure, IMPISH (IMP Information System Host):

- an introductory paragraph will describe the 'world' of software project management, the first modelised and implemented on IMPISH.

- secondly, we will define more precisely the types of information managed by the system,

- the following paragraph will identify the aims and the essential characteristics of this management by describing the languages for access to the system,

- paragraph five will show the architectural composition and the role of the main elements to finally justify the design and the use of the different tools chosen for the realisation of the system.

2. One use of IMPISH
(a data model for the management of software projects)

Computer assisted management of software projects, a branch of Software Engineering, presents several areas of major interest:

Software Engineering workbenchs are today accepted as tools which favourise the production of quality software at reasonable cost (5) (6). They are normally organised around an information system (7) (8) where the reference data is stored, checked and modified.

It is this information that the project leader wants to manipulate, either interactively, or by using management tools (planning tools, status reports...) (4) (9) (10).

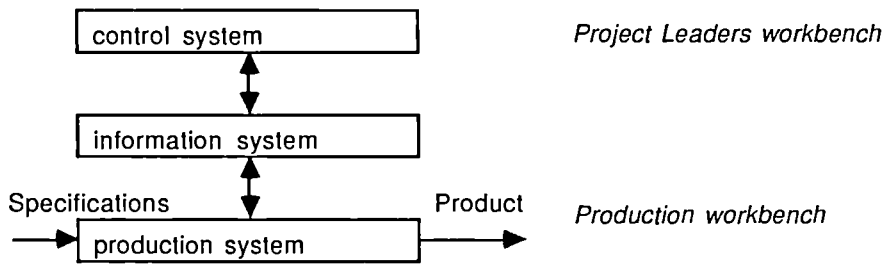


fig. 1 : Systemic Approach of the Organisations [20]

The 'data' handled in the scope of software production is extremely complex: work carried out by B.W. Boehm and his team has shown that in a similar 'ERR' expression, there can be as many as thirty entities involving more than 200 attributes and interconnected by some 170 relations. The mock-up of the Concerto information system has produced similar findings (8).

In a modelisation limited to management, we have evidenced some thirty entities and more than forty relationships uniting in excess of 100 attributes.

Moreover, our analysis of the management process (12) has shown the existence of numerous documents, either composed by the manager (at least twenty different types), or consulted by him (in fact more than thirty different types generated by the production environment).

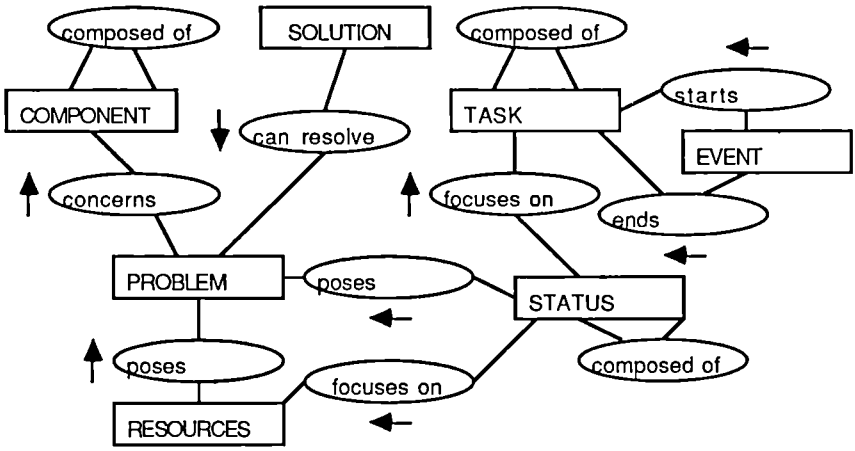


fig. 2 : Subset of the mode implemented in IMP Workbench

Software Quality Manual
Software Quality Assurance Plan
Task and Resource Plan
Progress Report
...

fig. 3 : Some documents producted by the IMP Workbench

Finally, a series of extremely detailed enquiries, in the form of interviews of management experts, has led us to the extraction of:

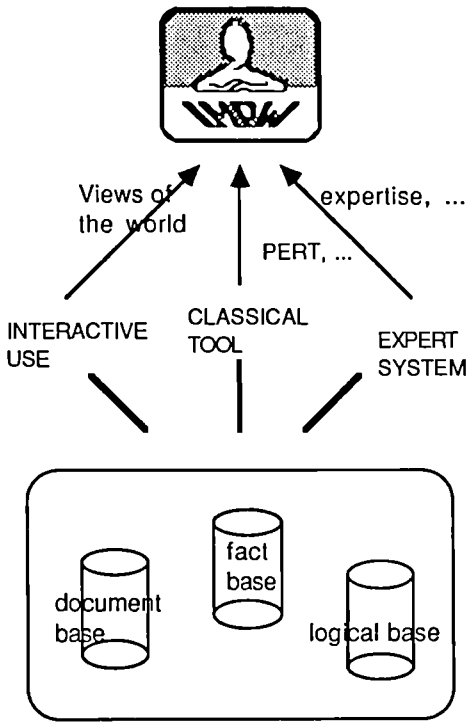
- Constraints

- Systematic Behaviour leading to a modification of data.

It is important to comment at this point that the constraints can affect the entity values, just as much as the conditions of existence of the documents, or even both at the same time:

For example, it is interesting to be able to add a document to the body of information which was used for its compilation and be able to signal its obsolescence in case of modification of the information.

The use of these Facts, Documents, Logical Behaviour (Rules), may be simultaneous. It therefore justifies the setting up of a software structure, purpose-built, allowing the modelisation of this information and its handling.



3. The information in IMPISH

Formally, we can classify the information managed by IMPISH in three logical groups, three 'bases':

- the facts, which we will show simply in the form of a t-uple, expressed under the form EAR (14)

- the non-factual constraints and logical rules, expressed simply under the form of Prolog,

We will gather these two sorts of information under the generic title "modelised information", as opposed to the following:

- the documents, or, more generally, digital information usually entered in the files (physical information).

3.1 The facts

In the first version of IMPISH the modelisation of the facts has been simplified by the use of the Codd concept, and the base concept will be the relationship. Implemented on a relational DBMS the facts will be "rows" of "tables".

They are accessible using traditional SQL-like commands.

The integration of these commands into the traditional programming languages makes them accessible in reading and writing by C procedures, for example, and therefore to tools written in this language.

We have also developed an interface which authorises their use in Prolog programmes (15).

3.2 The logical rules:

By this we mean, at the same time the expression of constraints (rules needing to be verified) and that of actions ('daemons') in Prolog, as well as operations for the handling of facts (insertion, deletion, modifications to t-uples).

The notion of Constraint needs to be clarified:

- A constraint limits the number of different ways that a t-uple be instanced from a table and may also introduce certain notions of coherence between tables.

- The definition of an "existential" constraint (the verification of which conditions the validity of a fact) can be expressed by one or more "operational" constraints (the verification of which conditions the validity of an operation or a fact - insertion, suppression -).

An analysis of work aimed at classifying these constraints (16) (17) and the taking into account of conceptual notions (keys, value domains...) enables us to define a certain number of "key words":

- The constraints of unicity (keys),

- The dependency by reference:

  If A is a 'resource',
  If B is a 'class of resource',
  Then 'A. type of resource' must equal one of the instances of 'B. class of resource key'.

- The functional dependencies:

  If B is a 'class of resource',
  Then 'B. number of available units' must equal the number of instances of resource R in which 'R. availability equals 'free'.

- The multivalue dependencies:

  If T is a 'task',
  Then 'T. date start'+'T. duration' must equal 'T. date end'.

- The value domains.

It should be noted that certain of these concepts are taken into account in the relational DBMS, or that their writing is made easier (key, value domain...). But the logical programmation enables us to extend appreciably the implementation of this notion of constraint.

Another contribution as a result of the integration of an inference engine to the DBMS is the definition of the 'daemons' on the relationships: they can be expressed by the activation of actions once an operation on the table has been validated.

We will see later how we have been able to define a definition language for these constraints.

3.3 The documents:

We will refine the definition of these by separating the documents tagged by an entity attribute (REF DOC) from those which tag a group of attributes belonging to different elements of the model, and constructed by one of the tools which may access the information system (TOOL DOC) (cf. fig. 4).

The documents are accessible through the research mechanisms of the DBMS, and their content, once tracked may be handled by no matter which appropriate tool (word processor for a document, editor or compiler, for a source code...).

In the case of tagged documents, a document instance is considered as an attribute (a "column value"). We shall call this special "column" a 'link'.

In the case of TOOL DOC a document type is considered as an integral entity - as a "physical document" type - which is added conceptually to the model.

Its attributes are:

'identifier'
'access path'
'type'
'version number'.

The access path is, in fact, the access path to a copy of the document (and not the document itself).

The 'type' is a generic semantic name, the value domain of which is defined for one application: Quality Plan, Meeting agenda...

The identifier is the current name of a given instance of a document of type 'type'.

The key of this entity is composed of 'identifier', 'type', and 'version number'.

For each type of document there will exist an instance "draft", corresponding to a syntactical frame emptied of all semantics, a standard instancing, the introduction of which is carried out at the moment the model is defined.

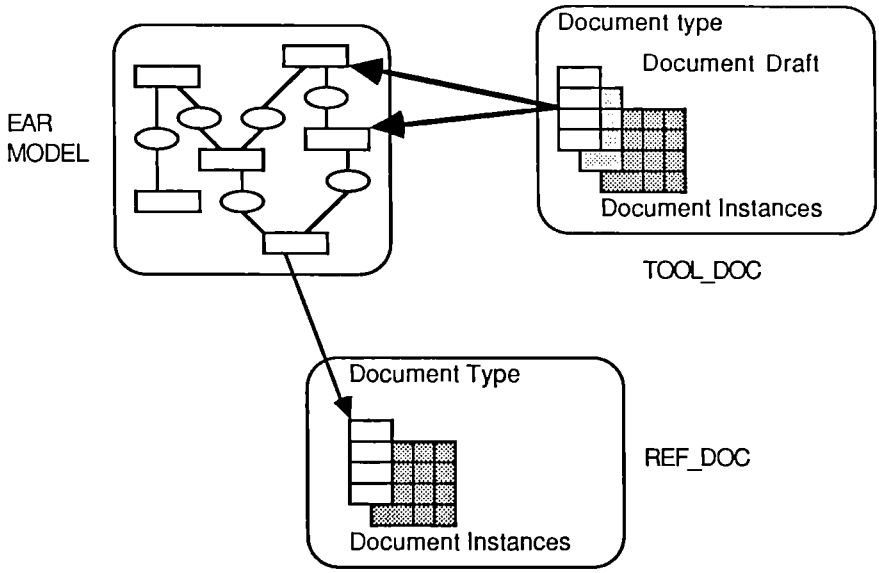


fig. 4 : The Links between the relational data base and the document base

3.4 The Management of Time with IMPISH

Three kinds of temporal information can be managed in IMPISH:

- The live informations which represent the current state of the base,
- The historical informations which deal with the past of the base,
- The simulated informations why are hypothetical and do not consist in shared references,
- On an earlier version, we applied these notions to the only facts managed by the DBMS and documents.

We plan to implement a temporal management of logical rules in the near future.

4. The Management of Information in IMPISH

The aims of IMPISH are to ensure:

- The definition of each of these data types and the corresponding models: it implements a data definition language which enables the definition of the EAR model, the constraints on the model, the model of the documents, the constraints on these documents, the 'daemons' acting on these models,

- The handling of this data, interactively by a user, but also automatically by the tools written in the procedural languages (C, Objective C) or declarational (Prolog): these tools (or assistants) can access to the knowledge of existing facts and modify them if the rights of access allow it,

- The autodocumentation of the commands and the model: this last service ensures, on line, a knowledge of sub-models and links (constraints...) the multiplicity of which makes the models global vision rapidly virtually impossible,

- The archival storage of bases (relational, logical and documentary),

- The possibility to construct with ease tools on an already-existing base,

- compatibility with SQL in the first version.

The realisation of these services is a result of the management of the coherence of the base of facts and documents, and the sharing of the information - notions of transaction, of rights of entry.

4.1 The DDL

The nature of the key words and the command syntax of the language will be very close to their counterparts in SQL (19). Their semantics will, however, be different particularly since what we call "base" is the union:

- of a traditional DBMS base (factual base),

- of a Prolog world (logical base) which contains the model,

- of a hierarchy of files (base documents).

Thus, the following commands, of which the syntax is SQL - like, are spread across these three worlds:

CREATE-DATABASE
CLOSE-DATABASE
DATABASE
DROP-DATABASE

Another command, ARCHIVE-DATABASE, has been added, to ensure that the base is recorded the instant it is activated, and to save the links which exist between "DBMS base" and "document base".

The commands which relate to the "tables" (cf relational tables) have he following behaviour:

CREATE-TABLE, which incorporates the keys and the links:

CREATE-TABLE table 1 (col1 col2) link1 link2 KEY col1 HD

Its activation gives rise to:

- the creation of a relational table,

- the adding of the relation to the Prolog model,

- the creation of a "unique index" on column 'col1' of the table,

- the creation of links, such as the adding of these to the Prolog models and the creation of a corresponding file space.

An optional key-word enables to define a column or table as historical, i. e. to be memorised in case of modification such a mechanism, based on the handling of SQL "VIEWS" will allow retrievals for any given period.

-DROP-NAME,

-RENAME-COLUMN, also pertaining on links

-ALTER-TABLE, which have a SQL-like behaviour spread across the different bases (logical and document)

-CREATE-DOMAIN the syntax of which is the following:

CREATE-DOMAIN (name of domain) ONTO (predefined domain)

PROGRAM (prolog program)

where the prolog program (18) defines a predicate the satisfaction of which confers to a value the quality of belonging to the domain.

Then come the commands which enable us to define and manage the different types of constraint:

-CREATE-REFERENCE (attribute) TO (attribute)

where an attribute is, in fact, a pair:

(table name) . (column name)

-CREATE DEPENDANCY (attribute) ON (attribute LIST)

ASSIGNING (assigning - LIST)

PROGRAM (prolog - program)

where:

(assigning - LIST)::= (attribute) = (prolog - variable)

```
                    (assigning - LIST) I

                    NIL;
```

The system manages the correspondence between the interested attributes (the first of the list of assignments depending on the others) and the prolog variables which "designates" them.

```
-CONSTRAIN (table name)
          WHEN (type of manipulation) (operational type)
     ASSIGNING (assigning list)
      PROGRAM (prolog program)
```

The type of manipulation may be INSERTION, DELETION or UPDATING, the operational type BEFORE or AFTER.

```
-CREATE-DAEMON (table name)
      WHEN (type of manipulation)
     ASSIGNING (assigning list)
      IF (condition) (other condition)
     THEN (action list)
      PROGRAM (prolog program)
```

where an action is a prolog goal, the realisation of which will lead to the manipulation of facts and rules in the system.

The structure of the prolog program which describes the conditions in the three previous commands is not trivial, and our work will lead to their formalisation.

-READ-CONSTRAINT (table name)

edits the list of all the constraints covered by 'table name' and sends back their internal number.

-DROP-CONSTRAINT (number)

removes the constraint of internal number 'number'.

The commands relating to the documents known "TOOL DOC":

- CREATE-DOCTYPE,
- RENAME-DOCTYPE,
- DROP-DOCTYPE, create, rename or remove the file spaces (directories) organised for the management of documents in IMPISH.

Finally, the DOL includes the SQL - like commands:

- GRANT,
- REVOK,

and a system command which allows the "compilation" of prolog programs (for example, expert systems in the field modelised in the base). It is this command, which when activated generates, at the same time, a new modified Prolog code and a set of rules declaring the file (pathname) as being interpretable by the metainterpreter associated to the current base:

- INTEGRATE (pathname).

4.2 The DML

The data manipulation commands offered by IMPISH start with:

- INSERT INTO (insert statement) (list of link assignments)

The "insert statement" is based on its SQL equivalent. However, (list of link assignments) is optional.

Here are some examples of the command grammar:

```
(list of link assignments) ::= nil | (list of link names)
                                     (list of link values);


(link value) ::= LINK (pathname) | VAL "(string";
```

This causes an extension of the command at the "base documents", by creating instances of documents tagged by the link name and containing either a character string (string), or the contents of the buffer file with the address (pathname)

- DELETE FROM (table name) (condition statement)

(qualification);

where

```
-(qualification) ::= WITH (name of predicate) ((list of terms));
```

This notion of qualification is an extension of the conditional expression (the SQL "WHERE" statement) which allows an operation to start only if the predicate "name of predicate", activated on the list of terms, is verified. This verification is executed by the Prolog inference engine.

You will notice that the removal is spread to the base documents, but the conditions of execution cannot affect the content or the address of a link.

- UPDATE (table name)

SET (update statement)

(condition statement)

(qualification);

Cf. DELETE above and the UPDATE SQL.

- SELECT PLUS (selection list) FROM (list of table names)

(condition statement) (order statement) (qualification);

where:

(selection list) ::= (select list) (select link list);

(select link list) ::= (select link) (select link list);

(select link) ::= (link name) | VAL (link name);

This SELECT gives access to the tagged documents by using the navigation mechanisms of the OBMS, either by returning a pathname, or by returning the content of the documents (VAL)

You will notice that the selection mechanism is extended by using the screening ((qualification) optional))

The occurrence of Keyword HISTORY enables the activation of selection on the historical data base thus giving the validity start date (VSD), the validity end date (VED) for a given fact. Reciprocally we will access the status of a fact at any given date using the "virtual columns" VSD and VED in the (selection LIST) or the (condition statement).

Such a feature is particularly useful with statistic or analogic tools.

Several commands allow access to "TOOL DOC":

- STORE-TOOL-DOC (pathname) (type of document) (id)

(dependency expression);

where:

(id)::= (string) | (string) (version number) | (string) LAST;

This definition of (id) allows the document to be memorised, either as the last version, or as the replacement of the version (number of version), or even as the replacement of the last version (LAST).

The (dependency expression), the grammar of which is not completely rigid allows us to define the degrees of dependency of the document with a certain number of instances of entities from the modelised base.

- DELETE-TOOL-DOC (type of document) (id*)

where:

(id*) ::= (string*) | (string*) (version number)

I (string*) LAST;

(string*) satisfying the specifications of the star convention of Unix.

This command destroys the document or documents concerned.

- GET-TOOL-DOC (type of document) (id*)

(selection of documents expression);

This command returns the documents selected:

- either by using the star convention,

- or by using (selection of documents expression) which allows access to documents which "depend" on instances of relations, themselves eventually determined by the combination of a "select statement" and a screening (qualification).

We will soon see a considerable evolution in these commands, brought about by object-oriented techniques which will allow us to return to a single structure (an object) the set of data linked to a document: date of creation, dates of handling, identification of handlers...

Finally, IMPISH offers commands for the use of classical transaction mechanisms of the OBMSs extended to the logical worlds and of documents (by keeping a log...):

- LOCK,

- UNLOCK,

- BEGIN-WORK,

- COMMIT-WORK,

- ROLLBACK-WORK.

Any errors arising during the use of IMPISH are managed by a mechanism of error treatment which signals in which base (relational, logical, or document) the errors occur.

5. The architecture and operating mode of IMPISH

The aim of IMPISH is to bring together the abilities of a DBMS and a logical programming language, not to rewrite one or the other. It was our wish to look upon the base components of our system as "black boxes".

INFORMIX*, Prolog II** and the FMS (File Management System) of UNIX*** are respectively the DBMS, the logical language, and the FMS of IMPISH.

The DBMS allows the memorising and handling of the facts, the SGF the documents. We will not elaborate on their particular functionality.

The logical language, is the medium of expression for the knowledge (essentially that of constraints, daemons and general model), and the opening to tools of "expert system" type.

A fourth component of our system, the "Translator", provides the interfacing of IMPISH with the tools of application following an object-oriented philosophy (cf.fig. 5).

Finally, an interactive handling module is being developed on the system.

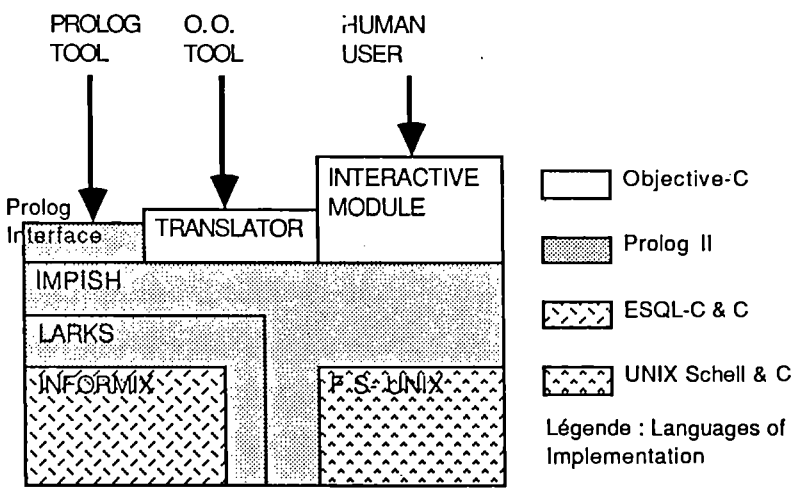


fig. 5 : Software Architecture of IMPISH

*INFORMIX: Relational Database Systems

**Prolog II: PrologIA

***UNIX: AT&T - Bell Laboratories

5.1 Prolog in IMPISH

_ as a knowledge medium:

The database model, the model of the base documents and of the constraints, links and daemons, are represented in the Prolog environment.

Furthermore, the adding of mechanisms (SL resolution, Forward reasoning...) to the base principles of Prolog increases the inference capacity.

- like the environment coupled to the DBMS:

We will not elaborate on the techniques used at this point:

The work we have done in this direction has led to the development of a prototype, LARKS (15) (Logic and Relational Knowledge System), in which several approaches (compilation, metainterpretation) have been implemented.

For the moment reduced to the porting of a single coupling procedure, we have continued to concentrate on the rewriting of the metainterpretation mechanism and the evaluation of the different methods. This will allow us to define an automatic process optimising the choice of the method for access to the database, in relation to the program and the considered Prolog goal.

The new functionalities offered are as follows:

A Prolog goal related to a table of the DBMS will naturally try to erase itself in the Prolog world, but in the case of a failure, the attempt at erasing will be extended to the t-uples of the table.

If 'task' is a table, the goal 'task(x,y,3)' will generate the backtrack on the set of triplets (x,y,z) where z=3 of the relationship 'task'.

However, it is still possible to limit the domain of erasing to the Prolog world alone by evaluating the goal through the predicate 'local':

'local(task(x,y,3))' provokes the resolution of 'task(x,y,3)' in the Prolog environment alone.

Conversely, the insertion, the deletion and the modification of Prolog facts relative to a DBMS table, by using the predicates 'assert db', 'update db' and 'delete db' is made possible in the base itself.

Finally, other possibilities (saturation with repercussion in the database...) are being investigated at this very moment.

We should note that the use of Prolog allows a simple implementation of the syntactic command analysis, the execution of which is assured by external predicates added to the language, which enable fine control of the errors.

5.2 The IMPISH Translator

A real interface to IMPISH for the tools developed and implemented in "Object Oriented" mode, this component receives messages (for a presentation of the principles and the O.O vocabulary, see (21) (22)). Those messages define:

- a LMD command including variables,
('receive':"select task.name from task where task.duration=*x")

- the list of these variables and the objects assigned to them,
('var':*x)

- the name of the class which will contain the result of the request to the information system.
('rep': Taskname)

For example:

(translatorObject 'receive': "select task.name from task where task.duration=*x" 'var':*x:durationObject 'rep':Taskname)
Objective-C Syntax (22)

It then deduces the veritable command, for example:

"select task.name from task where task.duration =4"

It launches its activation (towards Prolog) and recovers the result which it transforms into an instance of the response class (here "Taskname")

5.3 The interactive module

This tool is destined at the same time to the administrator of the information system and to the user who wishes to consult the status of this information.

For once it is merely specified, it will allow an interactive and guided interpretation of the DDL and the DML (limited depending on the nature of the user), and the activation of the tools for handling documents (editors, compilers...).

It will incorporate a users manual and above all a function of autodocumentation and interrogation of the model of the different bases of the system.

6. Conclusions

The IMP Workbench project demonstrates the interest there is in using IMPISH for the management of software projects.

However, although the results to-date are satisfactory, they require deeper study:

- the methods of modelisation and of acquisition of knowledge according to the three "works" considered,

- the evaluation and improvement of coupling procedures (metaprogrammation and optimisation of the mechanisms),

- the formalisation of the writing of the constraints and daemons,

- the definition of the dependencies between physical and modelised information,

all these points continue to be the object of research work, articles, and thesis.

The use of IMPISH in the near future for other applications, and the realisation of the interactive module will enable its evaluation to be completed.

Acknowledgements

This study has been carried out under the auspices of the project ESPRIT P938, IMP Workbench, partially founded by the Commission of the European Community, and involving ICL (GB), prime contractor, the CETE Méditerranée (F), NIHE (Ir), Imperial College (GB), and Verilog (F).

The authors wish to thank all those involved in this project, particularly L. Boi, B. terranova, and K. Sebti for their contribution to the specifications of IMPISH.

References

[1] BERNIER J. & al.
Convergence des bases de données et des systèmes experts
MBD No 5, December 86
AFCET

[2] GARDARIN G., SIMON E.

Bases de données déductives: Langages de règles et récursivité
Journées FIRTECH: Bases de données et Intelligence Artificielle - Paris, April 87

[3] CEE, ESPRIT Program: Integration of Logic Programming and Data Bases - Venice, December 86

[4] BOI & al.
General Specification and Design of IMP Workbench
ESPRIT delivery - P938 -1987

[5] ANDRE E., MOREAU B., ROUGEOT B.
Vers un atelier flexible et intégré: le projet CONCERTO
CONCERTO - Perros-Guirec, février 86

[6] PCTE: Technical documentation

[7] BOLOGNA M., ROMOLI C.
An ER Database for Software Engineering: the Portable Common Tool Environment Approach
PCTE ESPRIT Project

[8] ALLEZ F., BOI L. BOUBENIDER Y., HECKENROTH H.
A mockup of a CONCERTO Workbench Program information system using F1 formalism
BIGRE No 43-44 - July 1985

[9] ALLEZ F., BOI L., BOSCO M., BENOIT S., de la MOTTE COLAS Y.
The CONCERTO Workbench pilot station
CONCERTO - Perros-Guirec, February 86

[10] PIMS Information Manager
PIMS ESPRIT Project documentation

[11] PENEDO M.H.
Prototyping a Project Master Database for Software Engineering Environments
IEEE - 1985

[12] IMPW General Specification and Design
IMPW ESPRIT Project documentation - P938 -

[14] CHEN P.P.
The Entity Relationship Model: towards a unified view of data
ACM Transactions on Database Systems - March 1976

[15] BOI L., BOSCO M., GIBELLI M.
Logic and Relational Knowledge System
EEC, ESPRIT Program: Integration of Logic Programming and Data Bases - Venice, December 1985

[16] NGUYEN G.T., OLIVARES J.
SYCSLOG: système logique d'intégrité sémantique
TIGRE - R.R. IMAG No 26 - January 1985

[17] MIRANDA S., VINCENT C.
Promenade avec CAMPUS
MEO No 2 - January 1986

[18] Prologia: PROLOG II, Reference manual

[19] Relational Database Management System:
INFORMIX - SQL reference manual

[20] LEMOIGNE J.L.
Les systèmes de décision dans les organisations
PUF - 1974

[21] Xerox Learning Research Group
The Smalltalk-80 System
BYTE No 8 vol. 6 - August 1981

[22] COX B.J.
Object Oriented Programming: An Evolutionary Approach
Addison Wesley, August 1986

Project No. 315

Software development in RAISE

Chris George

STC Technology Ltd, London Road, Harlow, Essex CM17 9NA, U.K.

1 Introduction

This work is part of Esprit project 315 RAISE — Rigorous Approach to Industrial Software Engineering. We show how the RAISE specification language (RSL) is able to capture the functional requirements of a piece of software in an initial specification, and is also able to capture particular design decisions as the specification is developed into an implementation in some programming language. To do this RSL has two important features — it is *wide spectrum* and it allows specifications to be encapsulated in **structures**. The RAISE method then allows one to assert and then prove relations between structures, and in particular the *implements* relation.

This paper shows how such relations may be asserted and proved, and how the resulting structures and relations between them are held in the RAISE data model. While several RSL specifications are presented it is assumed that most of the details of RSL are explained in [3], a paper on RSL also being presented during this technical week. It is also hoped that RSL is sufficiently well designed for some understanding at least to be immediate!

2 Initial applicative specification

This paper uses a running example, an implementation of a set of values as an ordered tree, so that it can be searched reasonably efficiently. (The values will have a natural ordering or a key on which they can be ordered.) But from the outside it should not matter how the set is organised, and so our initial specification can be more abstract than an ordered tree. In fact we start with an ordering as a list. We will first need to define a structure presenting values and their ordering.

```
VALUE =
structure
  type
    Value
  value
    le: Value × Value → Bool where
      ∀a:Value,b:Value,c:Value sat
        le(a,a) ∧
        (le(a,b) ∧ le(b,c) ⇒ le(a,c)) ∧
        (le(a,b) ∧ le(b,a) ⇒ a=b) ∧
        (le(a,b) ∨ le(b,a))
end VALUE
```

This structure defines a type *Value* and a linear order *le*. We will use *VALUE* in the structure defining our set. This effectively gives us a parameterised structure — we can later substitute any type with

a linear order for *VALUE*. This might be integers ordered with '$\leq$', names ordered alphabetically, or records with keys which can be ordered.

We now define a set as an ordered list:

```
VAL LIST =
structure
  use VALUE
  type
    Ordered_list :: those seq:Value* sat is_ordered_list(seq) opaque
  value
    is ordered list (seq:Value*) :Bool ≜
        match seq with
          ⟨⟩ then true,
          ⟨*⟩ then true,
          ⟨x⟩^(seq1 as ⟨y⟩^*) then le(x,y) ∧ is ordered list(seq1)
        end omit,

    add_to_list (i:Value,mk-Ordered_list(seq:Value*))
      gives mk-Ordered_list(seq1:Value*) sat
        elems seq1 = elems seq ∪ {i},

    del from list (i:Value,mk-Ordered list(seq:Value*))
      gives mk-Ordered_list(seq1:Value*) sat
        elems seq1 = elems seq\{i},

    is_in_list (i:Value,mk-Ordered_list(seq:Value*)) :Bool ≜ i ∈ elems seq,

    empty_list: Ordered_list = mk-Ordered_list(⟨⟩)
end VAL_LIST
```

If we want *VAL LIST* to be an abstract definition we do not want the fact that we have used ordered lists as our way of holding sets to be visible. We therefore **opaque** the type definition of *Ordered list*. This has the effect of hiding the definition of *Ordered list*, but not the name, from outside the structure. Note that we could not hide the name as well as the type or we would not be able to express the types of the values which are also visible outside. Thus we know from outside that the type of *is_in_list* is some sub-type of *Value* × *Ordered_list* → **Bool** and that of *add_to_list* is some sub-type of *Value* × *Ordered list* → *Ordered list* so that we may use the value of an application of *add_to_list* in an argument of *is_in_list*. Thus making a type definition opaque changes the possible ways in which we are allowed to develop it. If *Ordered_list* were not opaque we would only be able to use lists to represent it. Having made it opaque we can use something else (and we in fact intend to use ordered trees).

Note that in the definition of *Ordered_list* we have used '::' instead of '='. If we define some type T by a definition of the form $T :: T1$ where $T1$ is some type expression, then we can distinguish values of type T from values of type $T1$, because there is a bijection **mk**-T automatically defined, of type $T1 \to T$, and this function can be used in pattern matching. Thus we are making T and $T1$ isomorphic instead of equal — to any value of T there corresponds a unique value of $T1$ and vice versa. Types which we wish to opaque must be defined in this way; opaquing is really like hiding the **mk**- function.

On the other hand, we do not want the function *is_ordered_list* to be visible outside at all, because it is irrelevant to our abstract view of sets. Hence we **omit** it, and it is totally invisible outside the

structure. Both types and values may be omitted, and are visible outside unless omitted.

2.1 Implementability

Having written down a structure like *VAL_LIST* we would like some confirmation that what we have written makes sense. More formally, we would like there to be at least one model of our specification. For types we must show the type is non-empty; values are defined as members of a type, and so the same approach suffices for them as well. Sometimes type existence is guaranteed immediately. For example, if we define a type using a type constructor, such as $A \times B$, then its existence is guaranteed if A and B exist. Thus the existence of *Value** is guaranteed immediately, assuming *Value* exists, since '*' (which creates a list type) is defined for all types. For types defined by a sub-type predicate, the sub-type is defined if the type being sub-typed exists and there is at least one value satisfying the predicate, i.e. if the predicate is not contradictory. Therefore *Ordered_list* exists if we can find lists of values which satisfy *is_ordered_list*. Since the empty list will do, the type clearly exists.

For the (function) value *is ordered list* we need to check that for any list *seq* of type *Value** the function will return **true** or **false**. The constructive nature of the algorithm, and the well founded recursion (the recursive calls are on shorter lists) assures us this is so.

For *add_to_list* we need to check that for any *Value* and any *Ordered_list* we can construct another *Ordered_list* whose elements are exactly those of the original list plus the new element. This is clearly possible — we could outline a constructive algorithm to do so. The other functions are similarly easy to check.

2.2 Properties of a structure

So what properties does this structure have, viewed from outside? For example, we would like it to be true that if we add an element to an *Ordered_list* value, and then check if it is there, the answer should be **true**, i.e.

$\forall$(i:Value,s:Ordered_list) **sat** is_in_list(i,add_to_list(i,s)) = **true**

If we could *unfold* the function calls, i.e. use the definitions, we could prove this immediately. However, we are not allowed to unfold values of opaque types, because to do so would expose the particular representation used. We might then be able to prove, and so rely on, properties true only of that representation. But the whole point of making types opaque is for them to be developed into other types for which properties true only for the original representation no longer hold. We need some properties, i.e. a theory of *VAL LIST* which is independent of its opaqued type. To do this we may define another structure *VAL_SET* as follows:

VAL SET =
structure
  use VALUE
  type
    Set_of_val
  value
    add_to_set: Value × Set_of_val → Set_of_val,
    del from set: Value × Set of val → Set of val,
    is_in_set: Value × Set_of_val → **Bool**,

```
    empty set: Set of val
  value axiom
    ∀(i:Value,j:Value,s:Set of val) sat
        is_in_set(i,add_to_set(j,s)) = (i=j ∨ is_in_set(i,s))
        ∧
        is_in_set(i,del_from_set(j,s)) = (i≠j ∧ is_in_set(i,s))
        ∧
        is_in_set(i,empty_set) = false
end VAL_SET
```

This RSL structure is much more like specifications in the style of *algebraic* or *axiomatic* languages such as OBJ, but it has the same overall shape as *VAL LIST*. We have a single type *Set of val* but this time there is no definition for it. We then define four values, but this time we want to define them in terms of each other. A convenient way to do this in RSL is to give only very loose definitions in the **value** section, and then to give one or more axioms in a **value axiom** section. Here the **value** section establishes nothing more than the base types of the four values (and the fact that the functions are total, since we have used '$\rightarrow$' rather than '$\xrightarrow{\sim}$'). Thus any values with these base types that satisfy the axioms which follow will satisfy this specification.

Since there is no type definition there is nothing to opaque and no problem about seeing the definitions from outside. Thus given our earlier property

```
∀(i:Value,s:Set_of_val) sat is_in_set(i,add_to_set(i,s)) = true
```

(expressed in terms of the function values from *VAL_SET*) we can prove it immediately, since

```
is in set(i,add to set(i,s)) = i=i ∨ is in set(i,s)
                             = true ∨ is_in_set(i,s)
                             = true
```

We will therefore take *VAL_SET* as our most abstract specification, and show that *VAL_LIST* is a development of it. In particular, we want to show that the theory of *VAL SET* is a sub-theory of that of *VAL_LIST* (or, equivalently, that all models of *VAL_LIST* are models of *VAL_SET*). We may then describe *VAL LIST* as an *implementation* of *VAL SET*. In RAISE we always use the term *implementation* in this precise, formal sense.

Note first of all that it is common to regard models as defined 'up to isomorphism', i.e. to disregard the particular names used for things. This makes it possible to use *add to set* for the value from *VAL_SET* and *add_to_list* for the corresponding value from *VAL_LIST*. It would be very confusing if we had to use the same names for both! We will assume the obvious correspondence of *X_set* from *VAL_SET* with *X_list* from *VAL_LIST*.

To show that *VAL LIST* is an implementation of *VAL SET* we show that the theory of *VAL SET* is a sub-theory of *VAL_LIST*, i.e. that statements true in *VAL_SET* are true in *VAL_LIST*. Statements true in *VAL SET* are either stated in it in definitions or provable from them, so we only have to check those actually stated. It is a trivial task to check the axioms. For example,

```
is_in_list(i,add_to_list(j,s)) = i ∈ elems add_to_list(j,s)
                               = i ∈ (elems s ∪ {j})
                               = i=j ∨ i ∈ elems s
                               = i=j ∨ is_in_list(i,s)
```

Note that if there were a non-opaqued type definition in *VAL SET*, such as

Set of val = Value-set

(where -set is a built in type constructor) we would have had to find or prove a similar definition in *VAL_LIST*, and as things stand the implementation proof would not go through. This shows how non-opaqued type definitions restrict the possible implementations.

2.3 Development of applicative structures

We have already created one example of a development, in showing *VAL_LIST* to be a development of *VAL SET*. We are now ready to give the structure *VAL TREE* which uses a tree representation for our sets.

```
VAL_TREE =
structure
  use VALUE
  type
    Tree :: [Node] opaque,
    Node = those (l,v,r):Tree × Value × Tree sat
                match l with
                  nil then true
                  mk-Tree(n) then le(max_tree(n),v)
                end
                ∧
                match r with
                  nil then true
                  mk-Tree(n) then le(v,min_tree(n))
                end omit
  value
    max tree (tr:Node) :Value ≙
              match tr with
                (*,v,nil) then v,
                (*,*,mk-Tree(r)) then max_tree(r)
              end omit,
-- min_tree is similar to max_tree

    add_to_tree (i:Value,tr:Tree) :Tree ≙
              match tr with
                nil then mk-Tree(nil,i,nil),
              mk-Tree(l,v,r) then
                  if i=v then tr
                  elsif le(i,v) then mk-Tree(add to tree(i,l),v,r)
                  else mk-Tree(l,v,add_to_tree(i,r))
                  end
              end,
-- del_from_tree is also defined in the obvious manner


    is_in_tree (i:Value,tr:Tree) :Bool ≙
```

```
            match tr with
              nil then false,
              mk-Tree(l,v,r) then
                i=v ∨ is_in_tree(i,l) ∨ is_in_tree(i,r)
            end,

    empty_tree:Tree = nil
end VAL_TREE
```

We now need to check the consistency of our specification, as with those defined previously. The existence of types *Tree* and *Node* is easy to check, and function definitions are all constructive with well-founded recursion.

The next step is to prove that *VAL TREE* is an implementation of *VAL LIST*. To do this we can adopt a strategy based on that used for VDM, as defined for example in [1]. For each opaque type in *VAL_LIST* we need to find a type in *VAL_TREE* such that there is a surjective *retrieve* function from the latter to the former. Thus in this case we need a function of type *retrieve: Tree → Ordered_list* that is a surjection (i.e. all possible values of type *Ordered_list* can be generated from some Tree). The retrieve function is easy to define — it generates a list by depth first left to right traversal of the tree. The ordering on the list comes from the ordering on the tree.

Having proved this retrieve function to be total and surjective we would check that for any non-opaqued types in *VAL LIST* there were equivalent definitions in *VAL TREE* (where *equivalence* means that we may use the retrieve function(s) in showing it). This does not arise in this case, and so we are left with checking the values.

For each visible value in *VAL_LIST* we must find an equivalent value in *VAL_TREE*, such that the type of the value in the latter is (using the retrieve function(s) as appropriate) a sub-type of the type of the value in the former. For example, we must show that under the retrieve function the type of *add to tree* is a sub-type of the type of *add_to_list*.

But it is also worth noting that since *VAL_LIST* implements *VAL_SET*, if *VAL TREE* implements *VAL_LIST* it will implement *VAL_SET*, since implementation is transitive. Furthermore, it is precisely the property that *VAL TREE* implements *VAL SET* that we are interested in — the view that users have is of the set properties, not any special data structures used to give efficient implementations. It will therefore suffice to show that *VAL TREE* implements *VAL SET*, which merely involves showing that *VAL_SET*'s axioms are true in *VAL_TREE*. This is a fairly simple task. (Showing that *VAL_TREE* implements *VAL_SET* does not of course establish any implementation relation between *VAL_TREE* and *VAL_LIST*.)

3 Imperative RSL structures

We would like to use our *Tree* definition to create an imperative tree structure, i.e. one that holds a value of type *Tree* in a state. Such structures are commonly called *objects*. We therefore create the following object:

```
VAL_TREE_OBJECT =
select
  operation
    add: Value ~⇒ write,
    delete: Value ⇒ write,
```

```
    is in: Value ⇒ Bool read,
    can_extend: Value ⇒ Bool read,
    empty: ⇒ write
from
structure
  use VAL_TREE
  variable
    tr: Tree := empty_tree
  operation
    add (i:Value) write tr:Tree where is in(i) ∨ can extend(i)
      is tr := add_to_tree(i,tr)
    end add,

    delete (i:Value) write tr:Tree is tr := del from tree(i,tr)
    end delete,

    is_in (i:Value) :Bool read tr:Tree ≙ is_in_tree(i,tr)
    end is_in,

    empty write tr:Tree is tr := empty tree
    end empty,

    can_extend: Value ⇒ Bool read
end VAL TREE OBJECT
```

3.1 Filters

VAL_TREE_OBJECT is defined by defining an imperative structure and then filtering out the values (and the type *Tree*) inherited from *VAL TREE*. Note that only the signatures of operations are given in a filter (and that we use double arrows like '⇒' for operation types to distinguish them from function types). Note also that variables are never exported from structures, and so the operation signatures do not mention the name *tr*. Hence the type *Tree* does not need to be exported.

3.2 Bounded objects

You will notice that *can_extend* is defined differently from the other operations — in fact only its signature is given. So why has this operation been introduced at all? The reason for its introduction is that it is likely at some stage that we will have to put some bound on the size of trees we can store, but do not yet know how to specify such a bound. Should it be the number of nodes, or the depth of the tree, or some combination of these? How will it be related to the size of values we wish to store? But if we do not specify any restriction, and make the *add* operation total, it is impossible to do a correct implementation when we do want to introduce a bound. It is therefore a good idea to introduce an operation when the imperative object is first formed which can be used in the pre-condition of operations which seem likely to (structurally) extend the variable's value. Since we only know the signature of this operation at present (i.e. it will read the state variable's value, plus the value it is proposed to add, and return a Boolean) we only specify it this far. Thus without an axiom it is simply more loosely defined than it would be with one. We can later restrict its definition by making it more explicit as we develop the structure.

4 Structures as parameters

We have already met this notion in parameterising our structures *VAL SET*, *VAL LIST*, *VAL TREE* with the structure *VALUE*.

We are going to implement our tree object using a notion of storage. To do this we will first define a general notion of storage as a mapping from locations to elements, without giving any notion of what locations and elements actually are. Hence we will have a parameterised structure for storage, and we can later instantiate one or both parameters as required. We first define the structures *ELEMENT* and *LOC*, each of which does no more than define a type name.

```
ELEMENT =
structure
  type Element
end ELEMENT
```

```
LOC =
structure
  type Loc
end LOC
```

We can now define a storage type, with some values, in the structure *STORAGE*.

```
STORAGE =
structure
  use
    ELEMENT
    LOC
  type
    Storage :: Loc -f-> Element opaque,
    Locs_in_use = those (loc:Loc,st:Storage) sat is_in_use(loc,st),
    Extendable = those st:Storage sat is_extensible(st)
  value
    is_in_use (loc:Loc,mk-Storage(st):Storage) :Bool ≜ loc ∈ dom st,

    assign ((loc,mk-Storage(st)):Locs in_use,v:Element) :Storage ≜ mk-Storage(st + [loc→v]),

    get ((loc,mk-Storage(st)):Locs_in_use) :Element ≜ st(loc),

    extend (mk-Storage(st):Extendable,v:Element)
      gives res:Locs_in_use sat
        ∃loc:(those x:Loc sat x ∉ dom st) sat
          res = (loc,mk-Storage(st ∪ [loc→v])),

    contract ((loc,mk-Storage(st)) :Locs in use) :Storage ≜ mk-Storage(st\{loc}),

    is_extensible (mk-Storage(st):Storage) :Bool ≜ {loc|loc:Loc sat loc ∉ dom st} ≠ {},

    empty store: Storage = mk-Storage([])
end STORAGE
```

Note that since the final storage implementation will be finite we have included functions to extend and contract storage, and a notion of extensibility.

4.1 Instantiation of parameters

In order to create an implementation of trees using storage, we want to specialise the notion of *Element* in storage to something that can be a tree. A suitable type is *Tree1* as defined in the next structure *VAL TREE EL*.

```
VAL_TREE_EL =
structure
  use
    LOC
    VALUE
  type
    Tree1 :: [Loc]
    Node1 = Tree1 × Value × Tree1
end VAL_TREE_EL
```

We now create the appropriate storage model by replacing *ELEMENT* with *VAL_TREE_EL* in the next structure, *VAL TREE_STORAGE*. This is done with the keyword **providing**. In general we can write *STRUCTURE* **with** *B* **providing** *A* provided *STRUCTURE* uses *A*, and *B* implements *A*. In this case we are providing for *ELEMENT* (which is indeed used by *STORAGE*), and *ELEMENT* only exports one abstract type *Element*. Thus all we need is a structure exporting at least one type. *VAL_TREE_EL* exports two, and so we could use either. We in fact need *Node1* to replace *Element*, and we must say so in the **fitting** clause that is part of the **providing** clause. Unless the names are the same we must include such fitting information in **providing** clauses.

Note that the storage structure used below in *VAL_TREE_STORAGE* is not *STORAGE* but

```
STORAGE
with VAL_TREE_EL providing ELEMENT
fitting Node1 for Element
```

which is a new structure, which exists only inside VAL_TREE_STORAGE. We therefore give it a name *ST* to distinguish it from STORAGE. Technically, the form '**use** *A*' means that the using structure shares the structure *A* with any other structure containing the same use clause. The form '**use** *ANAME* = *A*' means that the using structure has created a copy of *A*, with the name *ANAME* which is not shared with other users of *A*. When we are **providing** for a use we must always make a local copy since it is a new structure we are creating.

```
VAL TREE STORAGE =
structure
  use
    ST = STORAGE
         with VAL_TREE_EL providing ELEMENT
         fitting Node1 for Element
  type
    Connected tree = those tr:(Tree1 × ST.Storage) sat is connected(tr),
    Stored_tree = those tr:Connected_tree sat is_ordered(tr)
```

```
value

  start_of (start:Tree1,store:ST.Storage) :Tree1 ≜ start,

  store_of (start:Tree1,store:ST.Storage) :ST.Storage ≜ store,

  is connected (start:Tree1,store:ST.Storage) :Bool ≜
    match start with
      nil then true,
      mk-Tree1(n) then
        if ST.is in use(n,store)
        then let (l,*,r):Node1 = ST.get(n,store) in
                 is_connected(l,store) ∧ is_connected(r,store)
             end
        else false
        end
    end,

  is ordered ((start,store):Connected tree) :Bool ≜
    match start with
      nil then true,
      mk-Tree1(n) then
        let (l,v,r):Node1 = ST.get(n,store) in
          (l=nil cor le(max_tree1(l,store),v)) ∧
          (r=nil cor le(v,min_tree1(r,store))) ∧
          is_ordered(l,store) ∧ is_ordered(r,store)
        end
  end,

  max_tree1 ((start as mk-Tree1(n),store) :Connected_tree sat start≠nil) :Value
    let (*,v,r):Node1 = ST.get(n,store) in
      match r with
        nil then v,
        * then max_tree1(r,store)
      end
    end,
-- min_tree1 is similar to max_tree1

  add_leaf (i:Value,store:ST.Extendable) :Stored_tree ≜
    let (y:Loc,store1:ST.Storage) = ST.extend(store,(nil,i,nil)) in
      (mk-Tree1(y),store1)
    end,

  add to tree1
    ((i,tr as (start as mk-Tree1(n),store))
    :Value × Stored tree sat
      start≠nil ∧
      (is_in_tree1(i,tr) ∨ ST.is_extensible(store))) :ST.Storage ≜
    let (l,v,r):Node1 = ST.get(n,store) in
      if i=v then null
      elsif le(i,v) ∧ l=nil then
                 let (y,store1):Stored_tree = add leaf(i,store) in
```

```
                    ST.assign((n,store1),(y,v,r))
                  end
        elsif le(i,v) then add_to_tree1((l,store),i)
        elsif r=nil then
                  let (y,store1):Stored tree = add_leaf(i,store) in
                    ST.assign((n,store1),(l,v,y))
                  end
        else add_to_tree1((r,store),i)
        end
      end,
-- del_from_tree1 is defined similarly

    is_in_tree1 (i:Value,(start,store): Connected tree) :Bool ≙
      match start with
        nil then false,
        mk-Tree1(l,v,r) then v=i ∨
                  is_in_tree1(i,(l,store)) ∨
                  is_in_tree1(i,(r,store))
      end
end VAL TREE STORAGE
```

The aim of producing *VAL TREE_STORAGE* is that it should be an implementation of *VAL TREE*. However, it should be immediately apparent that while *add_to_tree* in *VAL_TREE* can be invoked for any (ordered) tree, *add_to_tree1*, the corresponding function from *VAL TREE STORAGE*, can only be called either when the integer to be added is already present or when the storage can be extended. Thus we would in effect be trying to implement a function with a more partial one, and our attempts would be doomed to failure. We therefore go on to define *VAL_TREE_STORAGE_OBJECT*, since we know that the partiality of adding elements was included in *VAL TREE OBJECT*.

```
VAL TREE STORAGE_OBJECT =
select
  operation
    add1: Value ⇒̃ write,
    delete1: Value ⇒ write,
    is_in1: Value ⇒ Bool read,
    can extend1: Value ⇒ Bool read,
    empty1: ⇒ write
from
structure
  use VAL_TREE_STORAGE
  variable
    tree: Stored_tree := (nil,empty_store)
  operation
    add1 (i:Value)
      write tree:Stored tree
        where is_in1(i) ∨ can_extend1(i)
      is
      match start with
        nil then tree := add_leaf(store_of(tree),i),
        * then tree := (start_of(tree),add_to_tree1(i,tree))
      end
    end add1,
-- delete1 is defined similarly
```

```
    is in1 (i:Value) :Bool read tree:Stored tree ≙ is in tree1(i,tree)
    end is_in1,

    empty1 write tree:Stored_tree is tree := (nil,empty_store)
    end empty1,

    can extend1 (i:Value) :Bool read tree:Stored tree ≙ ST.is extensible(store of(tree))
    end can_extend1
end VAL TREE STORAGE OBJECT
```

5 Development of objects

We want to show that *VAL_TREE_STORAGE_OBJECT* implements *VAL_TREE_OBJECT*. If we look at the signatures of the two objects there is the obvious correspondence between the types, variables and operations. Since the type *Tree* is opaque we can use a retrieve function in the same way as we did between the applicative structures *VAL_TREE* and *VAL LIST*. Thus we define

```
retrieve1 ((start,store):Stored_tree) :Tree ≙
  match start with
    nil then nil,
    mk-Tree1(n) then
      let (l,v,r):Node1 = ST.get(n,store) in
        mk-Tree(retrieve1(l),v,retrieve1(r))
      end
  end
```

in the obvious manner.

retrieve1 is a total function on *Stored_tree* by definition. To perform a proof similar to that used earlier we would also need to show that *retrieve1* is a surjection. This is, however, not obvious — we would need to show that we had enough storage to create as large a tree as we could with *VAL_TREE_OBJECT*. We have some degree of freedom in deciding how *can_extend* should be implemented — this was why it was left vague — but it is not clear how to proceed.

We instead appeal to the notion of a *simulation* as described in [2]. This is more general than the technique of a retrieve function, but still sufficiently strong to show that the object specifications are behaviourally equivalent, which is what we require. The proof is not included here but is straightforward.

It leads to a condition on *can extend* of the form

```
∀(i:Value,str:Stored_tree) sat can_extend(i,retrieve1(str)) ⇒ is_extensible(store_of(str))
```

Since *can_extend* was not given any definition we can regard this as merely giving an implementation constraint on it. This is perfectly safe since, there being no definition for *can extend*, users were not entitled to make any assumptions about it beyond the fact that it returns a Boolean value.

6 The RAISE data model

We have now created a number of structures and relations between them. It is clear that we need some way of holding all this information. At the trivial level the first requirement must be some means of relating structures to their names. Without this the use clauses in structures would be meaningless.

The second requirement is that there must be some means of storing and accessing the proof obligations and proofs of the implementation relations between structures. In fact we generalise from this slightly and say that we want to be able to record such relations even when we know that they are not implementations, but are related in some less strong way. Thus in the running example we know that *VAL TREE STORAGE* is not an implementation of *VAL TREE* because the add operation is more narrowly defined in the former, but it still might be useful to show that it is related in the sense that *VAL TREE* is part of *VAL_TREE STORAGE*'s history, and may even share some properties.

This leads on to another requirement. We hope eventually to implement structures in some programming language. We will later want to maintain these programs — to correct errors or change their behaviour. To do this properly we need to follow their development backwards, to find the appropriate level at which an error was made or a decision recorded that is now to be changed. Thus we must be able to record our development process in such a way that it can be followed in reverse.

There is also a need to be able to divide projects up into pieces that can be developed separately, while having some confidence that when these separate pieces are combined again the result will obey its original specification. This is why we need substitution to be monotonic with respect to implementation. But it also means being able to record which developments are related to each other, both to control the effect of changes and so that the development process can be followed in reverse.

Lastly, there is a need to record and access non-functional requirements, and reasons for developing things in certain ways. Thus each structure will be associated with an informal text, and so will developments.

This leads us to the following simplified description of the RAISE data model.

6.1 Descriptions

We firstly extend the notion of a **structure** slightly to include informal text and also formal properties — any properties of a structure we have proved or asserted may be recorded here. We call these extended structures *descriptions*. Descriptions may be named and accessed by their names.

6.2 Relations

Relations between structures are indexed by the names of their source and target descriptions. They contain a statement and possibly a proof (at some appropriate level of rigour) of the semantic relation being asserted. They also record whether the relation is an *implements* relation. We have informally created the following relations so far:

Source	*Target*	*Implementation?*
VAL LIST	*VAL SET*	*Yes*
VAL_TREE	*VAL_LIST*	*Yes*
VAL TREE	*VAL SET*	*Yes*
VAL_TREE_STORAGE	*VAL_TREE*	*No*
VAL_TREE_STORAGE_OBJECT	*VAL_TREE_OBJECT*	*Yes*

6.3 Developments

As well as descriptions and relations between them we need to record how a specification was developed, in terms of what descriptions were developed from what, what relations were used to justify the steps in this development process, and what developments are related to others. For this purpose we use the notion of a ***development***. Developments are named (so that they can be referenced), and have associated informal text so that comments on a development, its non-functional requirements etc., may be documented.

Developments consist of lists of development steps. Each development step records three things. The ***body*** names the description that is developed by the step. The ***contracts*** names the set of developments whose steps contain descriptions that are used in the body but are being developed separately. The purpose of this field is to record developments contributing to this one, so that when tracing back from a later implementation one can discover the history of its development. The ***view*** may be used to name a more abstract description that is implemented by the body. Its use will become more apparent in the creation of the development *SET_DEV* below.

6.3.1 SET_DEV

The first development we shall set up is that showing the development of the applicative set which is developed into first a list and then a tree. The development is called *SET DEV*, and is created by generating the development step

{|body=VAL_LIST,contracts={},view=VAL_SET|}

We then want to add to *SET DEV* a development step with body *VAL TREE*. What should its view be? We could make it *VAL_TREE* itself (an option we always have, as any description implements itself), or we could make it *VAL_LIST* since we have recorded the appropriate relation between *VAL_TREE* and *VAL_LIST*. But it would be best to make it *VAL_SET*, since this gives the most abstract view of *VAL TREE* for the reasons we discussed earlier — it is expressed axiomatically. We therefore append to *SET_DEV* the development step

{|body=VAL_TREE,contracts={},view=VAL_SET|}

to extend *SET DEV*. (Note that we do not insist on any relations between development steps. Thus the relation between *VAL_TREE* and *VAL_LIST* is not a pre-requisite to adding this step.)

This illustrates one reason for having views, rather than only a body and contracts at each development step. It allows us the option of merely showing that we are still implementing the same abstraction. When a view changes between development steps it is a sign that something has changed. Our next development step is a typical example of this.

We now want to extend the development to the imperative form. We have *VAL TREE OBJECT* and *VAL_TREE_STORAGE_OBJECT*, and a relation recording that the latter implements the former. We can therefore add the development step

{|body=VAL_TREE_STORAGE_OBJECT,contracts={STORAGE_DEV},
view=VAL_TREE_OBJECT|}

where *STORAGE DEV* is the development of the storage model, described in the next section. It is used here because *VAL_TREE_STORAGE_OBJECT* uses *VAL_TREE_STORAGE* which uses *STORAGE*, and because *STORAGE* is to be developed separately. Note that it is only such separate developments that are referenced in the contracts field, not all used descriptions. The change of view between this development step and the previous one indicates that something has changed, and in this case it is the form of the interface to the descriptions which are now imperative (i.e. they include operations).

6.3.2 STORAGE_DEV

We have only one description in this development so far, and so must use it as the body and view. We therefore create the single development step

{|body=STORAGE,contracts={},view=STORAGE|}

We can extend *STORAGE_DEV* separately from *SET_DEV*. If we are able to maintain *STORAGE* as the abstract view while developing the body to *STORAGE1*, say, then it will be possible for *VAL_TREE_STORAGE_OBJECT* (or some later development) to replace *STORAGE* with *STORAGE1* with confidence that its theory will not thereby change. This is because substitution is monotonic with respect to the implements relation. That is, representing the implements relation by $\sim$:

From VAL TREE STORAGE OBJECT $\sim$ VAL TREE OBJECT
and STORAGE1 $\sim$ STORAGE
and the fact that VAL TREE STORAGE OBJECT uses STORAGE
if we let VAL_TREE_STORAGE_OBJECT1 =
VAL_TREE_STORAGE_OBJECT
with STORAGE1 **providing** STORAGE
then we can deduce VAL_TREE_STORAGE_OBJECT1 $\sim$ VAL_TREE_OBJECT

References

[1] C.B. Jones, *Systematic Software Development Using VDM*. Prentice Hall International, 1986.

[2] T. Nipkow, *Non-deterministic data types: models and implementations*. Acta Informatica, 22, pp 629–661, 1986.

[3] E. Meiling, *A spreadsheet specification in RSL*. This volume.

Project No. 315

A Spreadsheet Specification in RSL - An Illustration of the RAISE Specification Language

Erik Meiling

Dansk Datamatik Center
Lundtoftevej 1C
DK-2800 Lyngby, Denmark

The main features of the RAISE Specification Language RSL are explained. The language is illustrated by an example showing pieces of the specification of a simple spreadsheet system.

1 Introduction

RAISE is an acronym for a "Rigorous Approach to Industrial Software Engineering". The aim of the RAISE project is to construct a method and a specification language, based on mathematics, for the development of software in industry, together with a collection of computer based tools supporting the specification language and the method.

Software development projects can be divided into a number of development stages, and RAISE covers the stages ranging from specification, via design, to implementation. All those stages in a RAISE software development are described using the RAISE Specification Language, RSL. Therefore, RSL must be able to describe a problem both at a very abstract level and at a level close to the programming language chosen for the project. The method for developing an RSL specification from the most abstract specification through intermediate levels to the implementation is described in another paper presented at the ESPRIT Technical Week 1987 (George [1]). The present paper concentrates on illustrating the various features of the RAISE Specification Language. Additional examples of RSL specifications can be found in George [1].

1.1 The Basic Concepts of RSL

In this paper, RSL is described using a specification of a simple spreadsheet system as an example. First, the basic concept of an RSL *structure* is introduced. Structures are the building blocks and abstraction units in RSL. Structures constitute the frame in which the RSL entities *types, values, variables, operations* and *processes* are defined.

RSL types can be defined by constructive type expressions similar to the domain equations found in the meta-language of VDM (the Vienna Development Method, cf.Bjørner [3], Jones [4]). Types can be used in the definition of named values. An important kind of value in RSL is the function. A function can be defined in three different ways:

- *Explicit* definition using the language of value expressions within RSL.
- *Implicit* definition defining a function by a predicate relating the function result and the parameter.
- *Axiomatic* definition in which the function is defined through algebraic axioms.

A structure may introduce a *state* through the declaration of variables, and different instantiations of the state of a structure can be created through copying of such a structure. The variables of a structure are not directly accessible outside the structure. Instead, manipulations of the state of a structure is performed through calls of *operations* defined in the structure. Operations can be thought of as "functions" with side-effects, and an operation can be defined explicitly using the statements of RSL or implicitly by a predicate.

Parallel activities are described through the introduction of *processes* in structures. RSL processes are based on CSP (Hoare [2]). Processes communicate through named channels. They can be defined explicitly using RSL parallel combinators or implicitly through a *failure assertion* which is a predicate describing the allowed sequences of communication.

2 Introducing the Spreadsheet Example

In this section we illustrate the basic elements of the RAISE Specification language. The example chosen to illustrate RSL is a specification of features of a spreadsheet such as Jazz and Lotus[1]. We will only sketch some of the features of a spreadsheet system in RSL, but we emphasize that each structure shown is *complete* in the sense that it follows the rules of RSL. It is considered important to be able to express partial knowledge about the solution to a problem, especially in the early phases of software development.

A spreadsheet can be conceived as a rectangualar diagram of entries, each entry addressed by a row and a column identifier. An entry may be empty, it may contain data (in the example integers and text) or it may contain a formula computing the value of the entry on the basis of other entries in the diagram. This description can easily be expressed in the following RSL structure:

[1] Jazz and Lotus are trademarks of Lotus Development Corporation.

```
SHEET_TYPES =
    structure
        use ENTRY_TYPES =
            structure
                type
                    Row_id,
                    Col_id
            end

        type
            Entry_id = Row_Id × Col_id
            Entry    = [| empty_entry : Unit,
                          number      : Int,
                          string      : Text,
                          form        : Formula |]
            Formula  = Sheet → Data
            Sheet    = Entry_id -f-> Entry
            Data     = Entry dropping form
    end
```

The *structure* is the building block and abstraction unit in RSL. The structure above contains only type definitions, but RSL structures can also be used to define *values, processes, operations* and *variables.* Structures are combined using the *structure operations* as explained in section 5. The type definitions will be explained in the following section.

3 Types in RSL

RSL types should be thought of as sets of values. The type definitions of *Entry_id, Entry, Formula, Sheet* and *Data* show how types can be constructed explicitly by type expressions (the right-hand sides of the equality signs). RSL is rich in providing type operators, and in this RSL follows the tradition of VDM.

The definition of *Entry_id* shows a type defined as a Cartesian product of other types. The type *Formula* consists of all functions from the type *Sheet* to the type *Data.* The type *Sheet* consists of all functions from a finite subset of the type *Entry_id* to *Entry.* The types *Entry* and *Data* are so-called *union types*, which can be though of as a set of values where each value has a tag indicating the kind of the value. The possible tags for values of type *Entry* are "empty_entry", "number", "string", and "form". Values of union types are written as shown by the following examples of values of type *Entry*:

```
[| number= 7 |]
[| string  = "house"|]
[| empty_entry |]
```

Unit is a predefined type used to indicate the absence of information contents. The three values shown above are also of the type *Data*, since this type consists of all the values of

Entry not having the tag *form*. Union types are often used in conditional constructs using the tags to choose among different cases. An example of this is shown later.

The definition of the types *Row_id* and *Col_id* in the inner structure *ENTRY_TYPES* is given without an explicit type expression. This means that nothing can be assumed about the construction of these types. The reason for embedding the types in an inner structure will become apparent in the subsequent section on structure operations.

Subtypes can be formed by attaching an arbitrary predicate to a type. Consider the type definition of sheet:

Sheet = Entry_id $\xrightarrow{f}$ Entry

Since the type *Sheet* only contains functions defined on a finite subset of *Entry_id*, this definition expresses that only finitely many entries are in use at any given time. Suppose we consider an implementation of spreadsheets in which only the non-empty entries are represented. This can be done by changing the above definition to

Sheet = Entry_id $\xrightarrow{f}$ (Entry **dropping** empty_entry)

Alternatively, the same effect can be achieved by defining a subtype in which we select the appropriate values by a predicate:

Sheet = **those** f: (Entry_id $\xrightarrow{f}$ Entry) **sat** [|empty_entry|] $\notin$ **rng** f

i.e. the value [| empty_entry |] is not in the range of any function in the type *Sheet*.

4 Values in RSL

Since RSL is rich in providing type operators, it follows that RSL is rich in the facilities offered for specifying values (data). In the previous section, we have seen how to construct Cartesian types, union types and function types. This section gives examples of other type operations as well as examples of the way in which values of such types are written.

The examples are given as value declarations in a structure. Such a value declaration consists of a name, a type expression and (optionally) an expression denoting the value in question.

```
EXAMPLE =
    structure
        value
            i     :   Int = 3
            ilist :   Int* = ⟨1, 13, 5⟩
            iset  :   Int-set = {x * x | x : Int sat 1 ≤ x ≤ 9}
            fct   :   Int x Int → Int = λ (x,y). x + y
            irec  :   {|f1 : Int, f2 : Int|}
            ixi   :   Int × Int = (3,6)

        value axiom
            irec.f1 = 3
    end
```

The value *i* is of the predefined type Int and is given the value 3. Notice that we are talking about named values, not variables. The value *ilist* is a list of integers, and it may be manipulated using the usual list operators such as head, tail and concatenation. In general, the use of type operators introduces a number of predefined operators on values of the type. The value *iset* is a set of integers, and its value is given by a *set comprehension* in which a predicate is used instead of explicit enumeration. *fct* is a function given a value through a function "constant", i.e. a lambda expression. Function values can also be specified in a more traditional syntax as shown in the next section. The identifier *irec* is of a record type (similar to record types of most programming languages) with field name *f1* and *f2*, but in the example its value is not given. Instead, an *axiom* is given stating that the field *f1* has the value 3. The value of the field *f2* is left unspecified. In such cases when more than one semantic model fits an RSL specification, we say that the specification is *underdetermined.* Underdetermined specifications are often used in the early stages of the software development, since it allows the specifier to leave possibilities open to be determined at a later development stage.

Splitting a value definition in two parts as it was shown for the value *irec* is often used in connection with the definition of functions. The part in which the name and the type is stated is called the *signature* part. An example of a function signature is

geq : Int × Int → Bool

The remaining part of the function definition can then be given in the axiom part. The use of signatures will help in highlighting the interface information of a structure. It can also be used in connection with operations and processes.

5 Structure Manipulations

In this section explain how structures are combined by the RSL structure operations. We use our example of the spreadsheet to show how the basic functions working on a sheet can be defined. We will define functions for inserting, deleting and evaluating entries.

```
BASIC_SHEET =
    structure
        use SHEET_TYPES
        value
            insert(sh: Sheet, ent_id: Entry_id, ent: Entry) : Sheet is
                sh + [ent_id → ent]

            delete(sh: Sheet, ent_id: Entry_id) : Data is
                sh + [ent_id → [|void|]]

            eval(sh: Sheet, ent_id: Entry_id) : Data is
                match sh(ent_id) with
                    [|form = f|]   then f(sh)
                    val            then val
                end
    end
```

The "use SHEET_TYPES" construct causes the entities introduced in the structure *SHEET_TYPES* to be included in the structure *BASIC_SHEET*. In the example, the entities are all types and they are used in the definition of the functions *insert, delete* and *eval.*

These three functions are defined *explicitely*, i.e. by an expression computing the result. The definitions of functions are just a special case of value definitions as shown in the previous section; RSL does, however, allow the more traditional syntax for function definitions shown above.

Consider the function *insert*. The result of an application of *insert* is a value of type *Sheet*, i.e. a finite function from entry identifications to entry values. The result is constructed using the infix operator "+", called function overwrite, such that the resulting function will only differ from the parameter function *sh* when called with the argument *ent_id*, in which case *ent* is returned. The definition of *delete* is similar. The definition of *eval* makes use of pattern matching as explained in a subsequent section.

Let us now return to the inner structure of SHEET_TYPES:

```
ENTRY_TYPES =
    structure
        type Row_id, Col_id
    end
```

By omitting type equations for *Row_id* end *Col_id* we have not committed ourselves to a particular representation of row and column identifiers. However, if we want to do more than trivial manipulations on values of these types, we need to commit ourselves further. In our example, we will use integers as row and column identifiers.

In order to express this commitment, we may choose to use an *extend* structure operation on *SHEET_TYPES* or we can make use of an *extend* structure operation on *SHEET_TYPES* or we can make use of a *substitution* structure operation replacing the

structure ENTRY_TYPES with a "more defined" structure. When using an extend operation, the commitment is expressed by adding axioms expressing the commiment. In the following, we will illustrate the second alternative, structure substitution. Using the structure

```
INT_ENTRY_TYPES =
    structure
        type
            Row_id  =  Int,
            Col_id  =  Int
    end
```

we can express the desired structure substitution in the following structure definition:

```
INT_SHEET_TYPES =
    SHEET_TYPES with INT_ENTRY_TYPES providing ENTRY_TYPES
```

Any structure appearing in a use-clause of another structure may be subject to structure substitution. The use of integers as row and column identifiers in *BASIC_SHEET* we could write

```
INT_BASIC_SHEET =
    BASIC_SHEET with INT_SHEET_TYPES providing SHEET_TYPES
```

or, more directly

```
INT_BASIC_SHEET =
    BASIC_SHEET
        with INT_ENTRY_TYPES providing SHEET_TYPES.ENTRY_TYPES
```

Structure substitution can be thought of as structure *parameterization* where the structure to be replaced plays the role of a formal parameter. The actual parameter must "provide for" the formal parameter, i.e. it cannot contradict any of the properties of the formal parameter. Therefore, the structure to be substituted is often referred to as a *requirement* stating the minimal set of properties to be expected.

6 Patterns

The function *eval* in the previous section made use of a match expression to compute its result. The match expression is a conditional expression in which the choice between the alternatives is made on the basis of *pattern matching*. Patterns are widely used in RSL specifications, and it is a powerful way to decompose data and express choices. In this section we will use list patterns and union patterns to illustrate this.

Consider the first pattern mentioned in the match statement of the function *eval*:

```
[| form = f |]
```

This pattern matches any value of type *Entry* having the tag *form*. In case of a match,

f becomes bound to the *Formula* component of the value (cf. the definition of the type *Entry* in section 2). The second pattern of *eval* is just

val

which matches any value. The identifier *val* becomes bound to the value matched. Pattern matching in match expressions is attempted in the order in which the patterns are given. From these examples we see that a successful match gives rise to a *binding* of the identifiers used in the pattern to the data components matched.

In our spreadsheet system, we need to be able to manipulate rows and columns of entries. We therefore extend the structure *SHEET_TYPES* with the following types defining rows and columns as lists of entries:

Row = Entry*
Column = Entry*

We can use these definitions to show various possibilities of list patterns. The patterns can be used in a context in which a value of type *Row* (or *Column*) is to be matched. The pattern

⟨ h ⟩

will match any row of length one and *h* will be bound to the entry in the row.

⟨ h ⟩ ^ t

This pattern will match any non-empty row (^ is the list concatenation operator). *h* will be bound to the head of the row and *t* will be bound to the tail.

x ^ ⟨ [| number = n |], t ⟩

will match any row in which the last element but one is a value with the tag *number*. The identifier *n* will be bound to the integer part of that tagged value.

Patterns can be augmented with a predicate giving a further constraint on the match:

⟨ [| number = x |], [| number = y |]⟩ ^ t where x = y

This pattern will match any row in which the first two entries represent identical integers.

The last example shows a function for determining whether two rows or columns have identical values. The function makes use of a Cartesian pattern to express the required matching condition. The pattern "*" matches every value.

```
eq(r1:Row, r2:Row): Bool ≙
    match (r1,r2) with
        (⟨ ⟩, ⟨ ⟩)
            then true,
        (⟨x1⟩ ^ r1, ⟨x2⟩ ^ r2) where eval(x1) = eval(x2)
            then eq(r1,r2),
        *
            then false
    end
```

7 Operations in RSL

In the spreadsheet example we have specified the central functions without being concerned with how to present the sheet to the user. The basic facilities for the user interface are defined in the structure below. The example shows how variables can be introduced in an RSL structure and how variables can be manipulated through the *operations* defined in the structure.

```
USER_INTERFACE =
    structure
        use SHEET_TYPES
        operation
            enter_entry   :   Entry_id × Entry ⇒ write
            get_entry     :   Entry_id ⇒ Entry read
            cut_entry     :   Entry_id ⇒ write
            paste_entry   :   Entry_id ⇒ write
            copy_entry    :   Entry_id × Entry_id ⇒ write
            show_sheet    :   read omit
        variable
            current_sheet :   Sheet := [ ]
            paste_buffer  :   Entry := || empty_entry ||
        operation axiom
            copy_entry(from_id : Entry_id, to_id : Entry_id)
                write current_sheet, paste_buffer is
                    paste_buffer := current_sheet(from_id);
                    current_sheet := current_sheet + [to_id → paste_buffer];
                    show_sheet
                end
    end
```

In the example, we have chosen to separate the definition of the operations in a signature part and an axiom part as explained earlier. Only the axiom part for *copy_entry* is stated, the other operations are left completely underspecified.

An operation can be thought of as a "function" with side-effects. The presence of side-effects is indicated by the use of "⇒" instead of the function arrow "→". Each operation states its access rights (read access or read/write access) to the variables of the structure; in signatures, however, only the access right to the variables as a whole is stated. Outside the structure, the variables are not directly accessible. Instead, the operations defined in the structure can be called to perform the required manipulations of the variables.

An operation may return a value, or it may only perform variable updates. The operation *copy_entry* above does not return a value (indicated by the absence of a type expression following the operation arrow). Its body is given as a sequence of RSL *statements*.

The above example also illustrates the use of the **omit** construct in the signature for the operation *show_sheet*. By using **omit** we express that the operation *show_sheet* is purely local to the structure *USER_INTERFACE* and consequently not an operation to be called by the user. This facility is a special case of a more general structuring operation in which

a structure can be defined from another structure by omitting and renaming entities.

The following example shows how we can create more than one spreadsheet using the structures already defined.

```
USER_INTERFACE2 =
    structure
        use U1 = USER_INTERFACE,
            U2 = USER_INTERFACE
        operation
            transfer_from_U1_to_U2(from_id : Entry_id, to_id : Entry_id)
                read U1, write U2 is
                    U2.enter_entry(to_id, U1.get_entry(from_id));
                    U2.show_sheet
                end
    end
```

The use clause of the above structure creates two named copies of *USER_INTERFACE* defined earlier. Since each of the copies contains a *Sheet* variable, we have effectively created two spreadsheets. The operation shown illustrates how these spreadsheets can be manipulated. Notice that the access clause contains the structure names instead of the variables names since variables are not visible outside the structure in which they are defined.

8 Implicit Definitions

All the value and operation definitions shown in the previous sections have been *explicit* definitions. This is somewhat atypical since *implicit* definitions are commonly used, especially in the early stages of development. In implicit definitions, a predicate is used to describe the desired behaviour of the value, operation or process being defined.

The following example shows an implicitly defined operation to be included in the structure *USER_INTERFACE*. The operation, *new*, takes no parameter and returns an entry identification as its result. The effect of a call of *new* is to create a new entry in the variable *current_sheet*.

```
new gives ent_id : Entry_id
    write current_sheet
    where
        ent_id ∉ rng current_sheet ∧
        current_sheet + [ent_id → [empty_entry]] = current_sheet'
    end
```

The predicate following the keyword **where** state the properties of the result and the transformation of the variables. Primed variable names refer to the value after the call of the operation, unprimed names refer to the value before the operation.

1 Processes in RSL

This section illustrates the use of RSL processes. The process concept in RSL is based on on CSP, Communicating Sequential Processes (Hoare [2]). Processes communicate values via named *channels*. A process has an *alphabet* giving the name, type and direction for each of the channels on which the process communicates. An alphabet consists of an *incomming alphabet* and an *outgoing alphabet*, each being a union type. The tag names are the channels and the type associated with a tag is the type of values which can be communicated on the channel.

A *communication* is a value in the alphabets of the processes involved. Communication takes place between processes running in parallel according to the following scheme: Any two processes whose alphabet define the same channel must engage synchronously in communication on that channel. If this is not possible, the two processes deadlock.

We illustrate the use of RSL processes by showing how multiple spreadsheets running in parallel can be handled. The reader may think of a system in which several spreadsheets are shown on the screen simultaneously and where commands to one spreadsheet can be given in parallel with the computation of formula etc. in other spreadsheets. To specify this situation, we replace the structure *USER_INTERFACE* shown earlier with the structure *SHEET_PROCESS* shown below.

For brevity, only the facilities for entering and copying entries are included. Instead of being operations, these facilities are specified as channels on which the parameters to the corresponding operations are passed. The channels *enter_cmd* and *copy_cmd* and their associated types are defined by the type *Command_alpha*, which is used as the input alphabet of the process *sheet_proc*.

The process *sheet_proc* manipulates the spreadsheet *current_sheet* defined in the structure *SHEET_PROCESS*. It is a loop accepting series of enter and copy commands from its environment.

```
SHEET_PROCESS =
structure
    use SHEET_TYPES

    type
        Command_alpha =
            [|enter_cmd : Entry_id × Entry, copy_cmd : Entry_id × Entry_id |]

    operation
        show_sheet:   read omit

    variable
        current_sheet  Sheet := [ ]
        paste_buffer   Entry := [| empty_entry |]

    -- continued on next page
```

```
process sheet_proc in Command_alpha is
    while true loop
        select
            input (ent_id, ent) from enter_cmd in
                current_sheet := current_sheet + [ent_id → ent];
                show_sheet
            end
        or
            input (from_id, to_id) from copy_cmd in
                paste_buffer := current_sheet(from_id);
                current_sheet := current_sheet + [to_id → paste_buffer];
                show_sheet
            end
        end -- select
    end -- while
end
end
```

This structure is now used in the structure *SHEET_PROCESS_SYSTEM* (shown on the next page) in which a copy is created for each spreadsheet. This is done in the "use SHEET = ..." clause creating a structure for each value in the type *Sheet_id.* In the example, three copies are created: *SHEET(sh1)*, *SHEET(sh2)* and *SHEET(sh3)*, thereby creating three processes having the names *SHEET(sh1).sheet_proc*,*SHEET(sh2).sheet_proc* and *SHEET(sh3).sheet_proc.* Since the type used as alphabet is also copied, each of these processes has a separate set of channels, e. g. *SHEET(sh1).copy_cmd.*

The communication between the user and the spreadsheets is handled by the process *command_proc.* Commands given by the user are modelled by having an input channel *request* on which a identification of spreadsheet and a command is recieved, expressed in the definition of the type *Keybord_alpha.* The output alphabet of the process *command_proc* is the union (created by the "or" type operator) of the sheet process alphabets as defined by the type *Command_alphas.* The command process is a loop receiving requests from the user and passing the command parameters on to the sheet process in question.

Finally, the entire system is expressed by the process *sheet_processes* in which all the spreadsheets run in parallel together with the command process.

```
SHEET_PROCESS_SYSTEM =
    structure
        use SHEET_TYPES
        use SHEET = [ id → SHEET_PROCESS | id : Sheet_id]
        type
            Sheet_id = [| sh1, sh2, sh3 |],
            Keyboard_alpha = [| request : Sheet_Id × Command_alpha |],
            Command_alphas = or {SHEET(id).Command_alpha | id : Sheet_id}

        process command_proc
            in Keyboard_alpha out Command_alphas is
            while true loop
                input req from request in
                    match req with
                        (sh, [| enter_cmd = cmd |]) then
                            output cmd to SHEET(sh).enter_cmd
                        (sh, [| copy_cmd = cmd |]) then
                            output cmd to SHEET(sh).copy_cmd
                    end
                end
            end

        process sheet_processes is
            parallel
                command_proc
            and
                parallel { SHEET(id).sheet_proc | id : Sheet_id}
            end
        end
    end
```

2 Experience with the Use of RSL

The aim of RAISE is to provide improved methods for the software producing industry. In order to ensure that the usability of RAISE in such industrial environments, three industrial trial projects are being carried out. One of these, undertaken by BBC Nordisk Brown Boveri A/S, is already in progress, and the remaining two projects, undertaken by ICL and BBC Nordisk Brown Boveri A/S, will start early next year. A structure editor has been developed for entering and editing RSL specifications. In the project already started, the process concepts will play an important role, since it includes the specification of protocols and communication. This project uses Modula-2 as its programming language. The experience gained from these industrial trials will provide input to an evaluation of RSL, and a revision of the language is planned in 1988.

Acknowledgement

The work reported in this paper represents the collective effort of the RAISE project teams. I would like to thank Jan Storbank Pedersen for helpful suggestions.

References

[1] George, C. W., Software Development in RAISE, this volume.

[2] Hoare, C.A.R., Communicating Sequential Processes, Prentice-Hall 1985.

[3] Bjørner, D. and Jones, C.B., The Vienna Development Method, Springer-Verlag 1978.

[4] Jones, C.B., Systematic Software Development Using VDM, Prentice-Hall 1986.

Project No. 432

GUIDE-LINES FOR BUILDING ADAPTABLE BROWSING TOOLS

J-L. Giavitto, Y. Holvoët, A. Mauboussin and P. Pauthe
Laboratoires de Marcoussis - CGE
Route de Nozay - 91460 Marcoussis - FRANCE

The aim of this paper is to outline well fitted concepts to enable adaptable and efficient browsing activities. This leads to the generation of a suitable environment frame from a data description, a data representation, tools and constraints declarations, and a communication specification. An existing prototype is briefly described.

1. INTRODUCTION

One of the METEOR investigation domains concerns the design of a prototype of an advanced development environment. Such an environment must support several formal languages to express requirements and to develop specifications. The development process requires the management of a large amount of data, made of complex entities and relationships, changing in time.

In this paper we present the language and the related architecture suited for building adaptable browsing tools. The emphasis has been put on concepts and techniques that can help in easily and quickly generating graphical user interfaces and databases well fitted to browsing into software engineering environments.

We first present the specifics needs of advanced software development environments. It is then proposed to meet those needs through a global approach providing an adaptable and powerful architecture for building environments. Finally, the implementation of a first prototype is presented.

2. NEEDS OF ADVANCED DEVELOPMENT ENVIRONMENTS

2.1. Supporting various activities

When developing large software products, environments should help efficiently in managing various kind of entities and relationships, from requirements engineering to software maintenance. An environment should then support various kinds of activities [2] [4] [9] such as: requirements engineering [14] [15], high level specification [3] [5] [8] [16], stepwise transformation from formal specifications to executable ones, prototyping and validating activities [20] [27], test data generation [6], project management, version management.

The various objects defined through these different activities are obviously not independent and have to be stored in such a way that every repercussion of any modification on one component be propagated over the other related components.

2.2. Supporting heterogeneous data

Related to the various activities, several formalisms can cohabit and must be supported at the same time in the environment. For instance, inside METEOR, several languages are under definition in order to express, in the most appropriate one, each of the various steps of the development process [5] [10] [14] [16]. Therefore, it is important to be able to express the existing relationships between components written in different specification languages. Moreover, it is also indispensable to express the relationships between components, design

decisions, pieces of documentation...

Let us take the example of a module of specification, independently of the specification language, it is involved in several contexts:

- as a component of one or more larger specifications,
- as a user of other specifications,
- as one of the various versions of this module,
- as a part of the development method, for instance the result of one of the transformation steps,
- by its relation to a part of another complex (structured) object: for instance, the associated documentation in natural language,
- by its relation to associated data sets for applicable tools,
- ...

Thus, such an object may be observed from several **viewpoints**, which show different facets of its complex reality. Viewpoints may themselves be represented by complex entities. They are dependent on the design method, the design language, the project management strategy...

2.3. Browsing activities

Searching relevant information at each step of the development process is a time consuming and, however, important activity, mainly to deal with the data complexity. A classification of browsing activities is given in [12]. Three categories emerge: search browsing, general purpose browsing and serendipity browsing.

Search browsing corresponds to queries with a precise aim: for example to get information on a specific module. General purpose browsing can be associated with a step by step exploration of a given context. Serendipity browsing corresponds to a "random" navigation through the data handled in the environment. It can be a trip through the different viewpoints attached to an object. This is especially useful for beginners to apprehend the application domain and the formalism used.

Browsing can imply an immediate access to the internal data representation, as well as complex computations. Therefore, browsing powerfulness relies on the expressivity of the requests and the efficiency of the data representation.

3. HOW TO SUPPORT BROWSING AMONG EVOLVING COMPONENTS

Browsing capabilities in advanced environment should be able to follow the evolution of the environment (data organization and tools). Moreover they must unify and simplify the various data manipulations.

The solution is to **generate** an appropriate environment frame from a description. The environment frame is mainly a well fitted user interface and database services for browsing, managing and maintaining information. Besides, this frame is designed to ease the integration of specific tools for the completion of the environment. The environment frame is described in Section 4 and the following paragraphs are concerned with the data organization.

3.1. Data organization

The ability to generate software development environments relies on a powerful language-independent model. In such a model, the browsing activities can be expressed in terms of navigation, independently of a particular instance (i.e. the actual structuring of a particular environment). Thus, browsing becomes a *syntactic activity* through the organization of data

without reference to the underlying semantics.

The model must enable the data structuring inherent in the specification language (i.e. the decomposition into entities and relations reflects the hierarchy and module decomposition provided by the language); it must also enable the data structuring due to the method for programming in the large (version management, documentation handling...).

In the approach reported in [17] the model describes a programming language in terms of a grammar and the navigation is done through a derivation tree. This approach does not satisfy our needs since it generates *programming* environments and not *development* environments.

On the other hand, a wider approach is obtained through the notion of *information systems*. But relational models used to structure usual databases are not powerful enough for describing the kind of information managed in a software development environment [21] [13]:

- In traditional databases or general object management systems the data descriptions are too **flat**. To make complex data palatable to the user, and to make the browsing efficient, a minimal condition is to **type** the data and organize them hierarchically.
- In classical databases there are numerous rather simple entities with few kinds of relations, while software engineering databases should handle complex objects with many kinds of relations [23]. Moreover these objects maintain complex interactions and only poor mechanisms are provided by classical databases for constraints propagation.

In the following paragraphs, we propose a model and we then present a set of requests to access data organized following this model and possible mechanisms for maintaining constraints.

3.2. The Graph Description Language

The data of the environment are structured according to a declaration made using the **Graph Description Language** (GDL). This declaration consists essentially of a hierarchy of node and graph definitions and is used to parameterize browsing commands and database management. It makes it possible to describe the components of an application in terms of a

```
nodes :: spec {  attributes ::  author:string, ... , signature:file, axioms:file, ... , compiled:boolean, ... },
        proc {  ...        },
        ...
graphs ::  specification { belonging nodes ::  spec, proc, param, draft, instantiate { }, ...
                           arrows ::  use { (spec, spec), (proc, spec), ... } , ...
                           edges ::  is an instance { (spec[0,1], instantiate[1]) },
                                     of { (instantiate[1], proc[0,1]) },
                                     using { (instantiate[+], spec) },
                                     parameterized by { (proc[+], param) },
                                     ...
                           attributes ::  ...
                           },
        documentation { belonging nodes ::        text {attributes ::  source:file },
                                                  picture {attributes ::  source:file }
                           arrows ::  include { (text | picture, text) }
                           },
        ...
viewpoints ::    expand { (spec, specification[0,1]), (proc, specification[0,1]), ... },
                 doc { (spec[0,1], documentation[0,1]), (proc[0,1], documentation[0,1]), ... },
                 ...
```

Figure 1 - Example of a partial LDG description

structure of typed attribute graphs.

An example of GDL description is given in fig. 1. It is a small part of the description convenient for a structuring adapted to specification development using PLUSS [5] [10]. Examples of graphs, structured according to this description, are shown in fig. 2; a more complete example, associated with the case-studies presented in [2] and [5], is shown in Fig. 5.

A **type**, declared using the GDL, associates a description ({...}) with a unique name:

- A *node description* is composed of a set of **attributes** (names and types: **string, boolean, file, graph, node**...).
- A *graph description* is made of sets of possible node-types (**belonging nodes**), possible links (**arrows** and **edges**) and **attributes** (names and types).
- An *edge* (resp. *arrow*) *description* consists of a signature: a set of pairs (resp. ordered pairs) of node-types with a possible constraint on the cardinality. It is *local* to a graph description.
- A *viewpoint description* consists of a signature: a set of pairs (node-type, graph-type) with a possible constraint on the cardinality.

Furthermore, a declaration in GDL automatically defines a more general graph-type, the "*FLAT-GRAPH*", defined by the union of all the graph declarations. This graph-type does not have any attribute and is needed to type the result of all possible requests (see Section 3.3).

The keywords **nodes, graphs** and **viewpoints** begin lists of node-, graph- and viewpoint descriptions. Several lists of this kind can exist; therefore, the node descriptions and the graph descriptions can be arranged in a modular and logical way. The defined nodes and graphs have a global scope, i.e. their names can appear in other descriptions. When the scope of a node description is restricted to a unique graph, the node description can be included in the **belonging-nodes** list (*instantiate, text* and *picture* in the example).

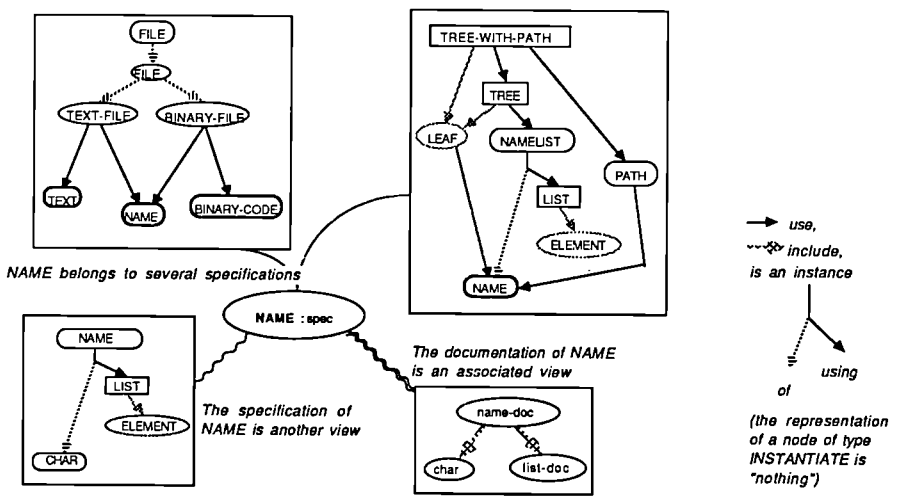


Figure 2 - Example of possible views

A cardinality constraint, noted *a[x]*, is the number of occurrences of a relation: adjacent to an entity of type *a* for an edge or a viewpoint, outgoing from a node of type *a* when *a[x]* is the left-hand part of the signature of an arrow, and incoming into a node of type *a* when *a[x]* is the right-hand part of the signature of an arrow. The following conventions are used: [+] for at least one occurrence, [n] for exactly n occurrences, [n-m] for a number of occurrences between n and m, [x,y...] for either x or y or... where x, y... are either n or n-m.

Several type declarations can describe the same information. Some of them may be more useful than others. An empirical rule for building practical representations is to associate a node with each conceptual or physical entity emerging from the methodology supported by the environment. The relationships can be described in GDL either as links (*use* in the example) or as nodes (*instantiate*) depending on the complexity of the relations. Therefore, graph structuring is obtained by determining the most important views involved during a specific application development, i.e. which types of nodes are related by which types of links in a given view (graph).

The structuring model derived from this language is an extension of the entity relationship model [24]. For example: it is possible to consider various graphs in which the entities are shared, graphs can be associated with entities, the attributes of an entity can directly take on values from the database (i.e. can be nodes or graphs)...

3.3. The access language

The kind of requests needed for efficient browsing activities are more complex than those provided by usual query languages. For example, the transitive closure of a relation is useful to handle dependencies between software components, and cannot be expressed using standard relational operators [19] (such kinds of extension are provided in [8]).

Access to data is done via the composition of access primitives and operators. The access primitives correspond to a direct naming of the entities. Naming convention are the following (where [] indicates a part that can be omitted if there is no ambiguity):

graph-name[:*graph-type*]
node-name[:*node-type*][@*graph-name*[:*graph-type*]]
entity-name.attribute where entity-name is the name of a node or a graph.

Operators are generic since they take a type as argument. They are split into two classes. In the first class, the type of the result is a type defined in the GDL description, including the type "*FLAT-GRAPH*". For example:

- *next* and *next** take a graph *g:G*, a set of nodes belonging to this graph and a set of link-types, and return a graph of type *G*. The result is a sub-graph of *g:G* whose nodes are immediate neighbours of the given nodes, connected by links of the link-types in the case of *next* and the transitive closure of the link-types in the case of *next**.
- *Selection* by predicates: $<(t_1, p_1), ..., (t_i, p_i), ..., (t_n, p_n)>$ *g:G* returns a graph of type *G* whose nodes are nodes of type t_i, belonging to *g* and satisfying the predicate p_i. Links of this graph are those connecting the selected nodes. The predicates p_i are built over the basic predicates (*belongs-to-a-graph?*, $\leq$, *empty-file?...)* and functions associated with the various types.
- Operations between graphs: *intersection, contraction* (of a sub-graph in a graph), ..., *union.* For example, the graph-type of the union of *g1:G1* and *g2:G2* is *G1* if *G1* $\equiv$ *G2*, *FLAT-GRAPH* otherwise.

In the second class, the type of the result is a multi-set of values of GDL-types. For example:

- *holding*(*node*) gives as result the set of graphs {g | *node* $\in$ g}.

- *associated-views(view, node)* gives as result the set of graphs attached to *node* by the viewpoint *view*.
- *associated-nodes(view, graph)* is the inverse of the previous one: the result is the set of nodes {n | *graph* ∈ associated-views(*view*, n)}.

When the result is a set of values, *iterators* can be used to apply another operator to such a result. For example:

- *ForEach(op, set)* is an iterator which applies *op* to each element of *set*.

The previous set of queries is well fitted to browsing among the data. Search browsing and general purpose browsing are well supported and they become syntactical activities. For example, requests concerning the *import* relation in COLD [16] as well as the *use* and *enrich* relations in PLUSS [5] can be handled by the same operators (*next**...). In general, browsing activities become simpler, e.g. serendipity browsing is mainly a traversal through the several viewpoints.

3.4. Tools and constraints

A development environment requires more than an efficient data organization. It must also include the control of the user's activity, i.e. the triggering of tools. In parallel with data browsing, the environment must enable an "activity browsing" to help the user to choose the tool he can trigger according to the current development step. Therefore, tools are described as partial functions that can be applied to nodes, graphs, links or attributes. They are defined by a signature and an applicability predicate; for instance, the symbolic evaluator only works on compiled specs; the tool description can be:

tools :: eval **{** (S:spec –> file) **if** S.compiled = true **}**

It is then possible to only propose the tools applicable at a given development step.

The database evolves after each tool activation which may lead to a temporary incoherent state. In order to return to a coherent state, the data management part of the environment frame provides ways to define constraints over the data and to maintain them. Various mechanisms exist to maintain the constraints:

- The typing, due to the GDL declaration, ensures basic constraints on the creation and destruction of entities. For example, such an operation is forbidden if it transgresses a cardinality constraint.
- A language enables the expression of some functional dependencies between objects via their attributes. For example, the maintenance of the compiled attribute of a spec is related to the use link; to express that a spec can be compiled when the used spec are themselves compiled we can state:

  N:spec use M:spec => N.can-be-compiled = AND (M.compiled)*

  (*AND** iterates the conjunction for every instance of *M* linked to *N* by a *use*).
- To avoid collisions and to allow the accesses only to coherent parts of the database, tools can obtain an exclusive access to the required sub-graph.
- Tools, themselves, are assumed to maintain some constraints. For instance, instead of maintaining the compiled attribute of a spec by the expression of a functional dependency, the spec editor can propagate the false value through the using specs (with the *next** request on the use link).
- Finally, the user can maintain constraints by hand.

4. ARCHITECTURE OF THE ENVIRONMENT

The structure of the environment is based on a customer-server model and has been designed to ease the integration of new application-dependent tools. The fixed part of the environment,

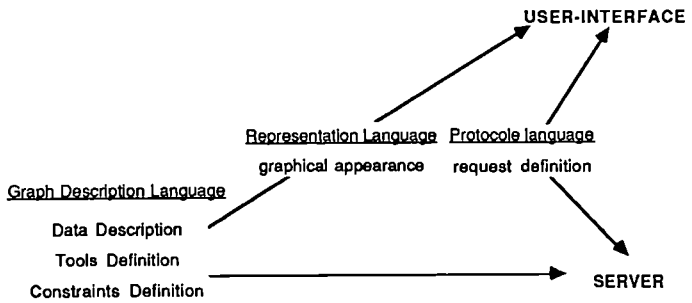


Figure 3 - Generation mechanism of the environment frame

(i.e. the application-independent structure), is parameterized by the GDL description. It is the frame in which application-dependent tools are integrated. It fulfils three main functions: The dialogue with the user, for navigation and driving purposes, is implemented by the **user-interface**. The data management is implemented by a unique process: the **Server**. Finally the **communication** between the tools, the user-interface and the Server is implemented by a package of library functions. The user-interface and the Server can be seen as a shell encapsulating the tools. The drawing of fig 4 displays a schematic view of a multi-users distributed access to the environment.

The environment frame is generated from descriptions expressed with several languages (fig. 3):

- the GDL declaration defines the data organization,
- the Representation Language (RL) declaration defines the graphical appearance of the GDL-types instances,
- the constraints and the tool definitions are expressed in the same formalism as the GDL declaration,
- the communication description is expressed with the Protocol Language (PL) [7].

4.1. The Server

Its main functions are:

- management of the data handled in the environment: creation/destruction/updating, storage/retrieval, constraints propagation...,
- management of the schemes which structure and parameterize the environment,
- decoding and interpretation of the interface and tools requests,
- management of concurrent accesses.

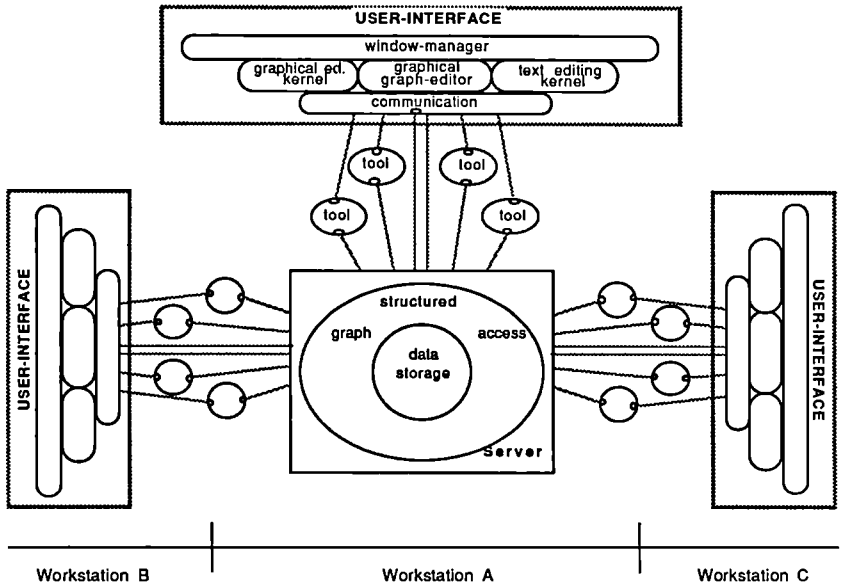


Figure 4 - Architecture of the environment with distributed accesses

The Server has a layer structure and some layers are automatically generated from descriptions. For instance, the top-level part of the request interpreter is generated by the compilation of the communication description in PL (see Section 4.3). Basic storage/retrieval functionalities are generated from the GDL declaration. Over these storage/retrieval functions, there is an application-independent layer which enables the graph manipulations required to implement the queries presented in Section 3.

Implementing the Server as a unique process gives several advantages:

- it simplifies data integrity handling and access control,
- it provides a uniform data access interface and facilitates tool integration,
- it provides an efficient tool coupling mechanism (the tools share the data representation) to build tools combinators (like for example pipes in UNIX).

4.2. The user interface

This part is application-independent and is the privileged way to interact with the environment. It is window and menu based, and provides graphical functionalities.

- Graphical functions provide graph display and edition. Thanks to them, the user can update data stored by the Server and browse the result of a request. In addition, facilities are available to apprehend large amount of data: graphical filtering and abbreviation, graphical holophrasty, various algorithms of standard graph display, history of commands...

- Alphanumerical edition capabilities are used to edit interactively small amounts of data, like attributes or commands. The classical edition of files is provided by the integration of specialized editors into the environment, following the tool integration mechanism.
- Command interpretation allows an interactive access to the Server.
- Display services are available for "non highly interactive" tools, i.e. basic standard functions for tools which don't need too much sophisticated interaction.

All the functions related to the screen for displaying purposes are parameterized. Each user can set parameters through declarations made in the Representation Language (RL). For example, the icon associated with a node-type has a basic form that can be a square, a circle..., in bold, in double pen... or even can be painted with an icon editor.

4.3. Tools integration and cooperation mechanism

Tools, like customers, get their resources from the environment through the user interface and the Server. Updating and retrieving data, as well as interaction with the user, are performed by request exchanges. Thanks to this architecture, the integration of tools is easy because the interaction between them and the rest of the environment is only made by exchanging ASCII messages. Furthermore, it allows distributed accesses to the centralized Server.

The messages are formally described in a Protocol Language [7] which defines the syntax of all the possible requests. Each request is composed of the syntactic form of the sent and received parameters and of all the possible errors. The sent and received parameters are declared in terms of regular expressions on the basic types such as: string, digit... The compilation of a source written in PL produces a package of functions allowing symbolic manipulations of the requests.

5. EXISTING PROTOTYPE AND FUTURE DEVELOPMENT

The current state of the GDL enables, in the same description, the expression of both the data

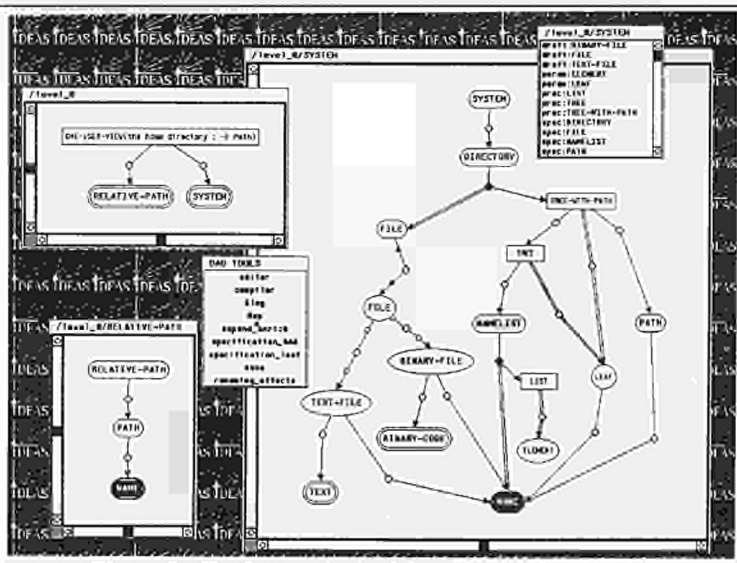


Figure 5 - Hard copy obtained from the existing prototype

structuring and the related graphical representation. For the moment, it provides only one kind of node expansion (viewpoint). The main purpose of this version was to point out a convenient formalism for representing the inherent structure of various applications. Demonstration examples (cf. fig. 5) have been implemented for several specification languages: PLUSS [5] [10], COLD [15] [16], ERAE [14], ALGRES [8]. For these examples, the implementation of the Server is simply build upon the UNIX file system.

The current version of the user interface relies on the UFO [1] C pre-processor for the internal management of the displayed graph. NEIGE [25] and LUGE [26] are the graphical libraries used for the screen management and the mouse interaction. The prototype runs on SUN 2/3 work-stations, under UNIX, with either black and white or color display.

The planned extensions are: the implementation of the entire GDL, the extension of the graphical capabilities of the user interface, the implementation of constraints maintaining, and the use of this environment to provide a powerful graphical environment for Slog [27].

Powerful and expressive models for software engineering databases, as well as real and efficient management of constraints, are still going on researches inside METEOR and in other projects (GRASPIN, [11], [17], [18]). From some points, our approach is comparable to the approach followed in the design of PCTE [22] or CONCERTO [11]. This approach is based on the concept of "shell" (common tool environment, "structure d'accueil", environment frame). Our generation of the environment frame is easily made through high level description languages. The implementation of the Server could be done using the OMS of PCTE.

REFERENCES

[1] M. Beaudouin-Lafon, *UFO User's Manuel*, Preliminary draft, LRI (Bat 490) University Paris XI, June 1986.

[2] G. Bernot and P. Pauthe, *Using IDEAS - A Scenario*, ESPRIT Project 432, Case studies, Draft, Meteor/t10/CGE-LRI/UIS.1, April 1987.

[3] J.A. Bergstra, J. Heering and P. Klint, *ASF – An Algebraic Specification Formalism*, Centre for Mathematics and Computer Science, Report CS-R8705, January 1987.

[4] M. Bidoit, *Experimentation of the ASSPEGIQUE Specification Environment*, METEOR Report, Task 11, November 1985.

[5] M. Bidoit, M.C. Gaudel and A. Mauboussin, *How to Make Algebraic Specifications more Understandable? An Experimentation with the PLUSS Specification Language*, ESPRIT Project 432, METEOR/t2-t10/CGE-LRI/UAS.1, March 87.

[6] L. Bougé, N. Choquet, L. Fribourg and M.C. Gaudel, *Test Set Generation from Algebraic Specifications using Logic Programming*, Journal of System and Software, 1986.

[7] O. Bourdon, H. Chatelier and Y. Holvoët, *Le Langage de Protocole (PL)*, Laboratoires de Marcoussis CGE, Internal Report, September 1986.

[8] S. Ceri, S. Crespi Reghizzi and L. Lavazza, *Extended Relational Algebra (ERA): Data Structures and Operations*, ESPRIT Project 432, METEOR/t2/TXT/1, May 1985.

[9] CGE-LRI-UoP team, *A Proposed List of Components for a Specification Development Environment*, ESPRIT Project 432, Case studies, Draft, METEOR/t11/CGE-LRI-UoP/SDE3, November 1986.

[10] M.A. Choquer, M.C. Gaudel, Y. Holvoët and A. Mauboussin, *A Concrete Syntax for PLUSS*, ESPRIT Project 432, METEOR/t10/CGE-LRI/1, March 86.

[11] CNET Lanion, *CONCERTO, Atelier de Logiciel*, Journées Concerto, Technical Presentation, February 4-6, 1986, Palais des congrès, 22700 Perros-Guirec, France.

[12] J. F. Cove and B. C. Walsh, *A Taxonomy of browsing*, Working Paper, University of Liverpool, Department of Computer Science, Chadwick Building Liverpool L69 3BX, April 1985.

[13] A. Doucet and M.C. Gaudel, *Bases de Données et Génie Logiciel: vers l'Intégration des Outils de Développement de Logiciel*, LRI, University of Paris XI, Journées Bases de Données de l'AFCET, La Rochelle, September 30-October 1, 1986.

[14] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, A. Rifaut, E. Stephens and F. Williams, *Model Components for Requirements Engineering*, Final report for METEOR Task 1, AT&T and Philips Telecommunications (Brussels), Philips Research Laboratory (Brussels), C.O.P.S. Computers Ltd (Dublin), September 1986.

[15] L.M.G. Feijs, J.H. Obbink and J. Hagelstein, *A Process Reference Model for Requirements- and Design Engineering*, METEOR/t6/PRLB-PRLEE/1, Philips Research Laboratories Eindhoven and Brussels, November 1986.

[16] L.M.G. Feijs, H.B.M. Jonkers, C.P.J. Koymans and G.R. Renardel de Lavalette, *Formal Definition of the Design Language COLD-K*, Philips Research Laboratories and Department of Philosophy, University of Utrecht, April 1987.

[17] J. Heering, G. Kahn, P. Klint and B. Lang, *Generation of Interactive Programming Environments*, CWI Amsterdam The Netherlands, INRIA Sophia-Antipolis & Roquencourt France, ESPRIT'85: RESULTS AND ACHIEVEMENTS, Elsevier Science Publishers B.V. (North-Holland), © The Commission of the European Communities, 1986.

[18] H. Horgen, *TOOL USE: An Advanced Support Environment for Method-driven Development and Evolution of Packaged Software*, Informatique Internationale S.A. Toulouse, ESPRIT'85: RESULTS AND ACHIEVEMENTS, Elsevier Science Publishers B.V. (North-Holland), © The Commission of the European Communities, 1986.

[19] S. Horwitz and T. Teitelbaum, *Generating Editing Environments Based on Relations and Attributes*, ACM Transactions on Programming Languages and Systems, Vol.8, No.4, October 1986, Pages 577-608.

[20] H. Hussmann, *Rapid Prototyping for Algebraic Specifications – RAP System User's Manual*, 2nd Edition, Universität Passau, February 1987.

[21] P. Pauthe, *A Survey on Databases for Software Engineering*, ESPRIT Project 432, METEOR/t7/CGE/SDSE, September 1986.

[22] PCTE Project Report, *PCTE: A Basis for a Portable Common Tool Environment*, ESPRIT'86: RESULTS AND ACHIEVEMENTS, Elsevier Science Publishers B.V. (North-Holland), © The Commission of the European Communities, 1987.

[23] M.H. Penedo and E.D. Stuckle, *PMBD-A Project Master Database for Software Engineering Environments*, 8th. International Conference on Software Engineering, London, August 1985.

[24] P. Pin-Shan Chen, *The Entity-Relationship Model — Towards a Unified View of Data*, Massachusetts Institute of Technology, ACM Transaction on Database Systems, Vol.1, No.1, March 1976, Pages 9-36.

[25] P. Raymond, *NEIGE: Un Noyau d'Environnement Intéractif et Graphique Elémentaire*, Laboratoires de Marcoussis CGE, Internal Report, January 1987.

[26] P. Raymond, *LUGE: Une Librairie d'Utilitaires Graphiques Extensible*, Laboratoires de Marcoussis CGE, Internal Report, January 1987.

[27] *Slog 1.1 User's Manual*, Laboratoires de Marcoussis CGE, August 1986.

Project No. 390

Formalisation of Developments: an Algebraic Approach

Bernd Krieg-Brückner

FB3 Mathematik und Informatik, Universität Bremen
Postfach 330 440, D 2800 Bremen 33, FR Germany

In the context of ESPRIT Project #390, PROSPECTRA (PROgram development by SPECification and TRAnsformation), a uniform treatment of algebraic specification is proposed to formalise data, programs, transformation rules, and program developments.

1. Introduction

Various authors have stressed the need for a formalisation of the software development process: the need for an automatically generated transcript of a development (history) to allow replay upon re-development ("maintenance") when requirements have changed, containing goals of the development, design decisions taken, and alternatives discarded but relevant for re-development [1]. A development is thus a formal object that does not only represent a documentation of the past but is a plan for future developments. It can be used to abstract from a particular development to a class of similar developments, a *development method*, incorporating a certain strategy. Approaches to formalise development descriptions contain a kind of development program [1], regular expressions over elementary steps [2], functional abstraction [3], and composition of logical inference rules [4].

In Transformational Development [5-7], the approach taken in the PROSPECTRA project [8,9], an elementary development step is a program transformation, the application of a transformation rule that is generally applicable; a particular development is then a sequence of rule applications. The question is how to best formalise rules and application (or inference) strategies.

The approach taken in this paper is to regard transformation rules basically as equations in an algebra of programs (chapters 2, 3), to derive basic transformation operations from these rules (chapter 4), to allow composition and functional abstraction (chapters 5, 6), and to regard developments as (compositions of) such transformation operations (chapter 7). Using all the results from program development based on algebraic specifications we can then reason about the development of transformation programs or developments in the same way as about programs: we can define requirement specifications (development goals) and implement them by various design strategies, and we can simplify ("optimise") a development or development method before it is first applied or replayed.

2. The Algebra of Programs

2.1. The Algebra of Data and the Algebra of Programs

In the PROSPECTRA project, loose algebraic specifications with partial functions and conditional equations [10] are used to specify the properties of data and associated operations in PA^{nn}dA-S, the PROSPECTRA Anna/Ada specification language [8,9]. For example, the fact that, for the mathematical integers INT, (INT, *, 1) is a monoid could be specified as in (2.1-1).

Similarly, we can define the Abstract Syntax of a programming language such as PA^{nn}dA-S by an algebraically specified Abstract Data Type: trees in the Abstract Syntax correspond to terms in this algebra of (PA^{nn}dA-S) programs, non-terminals to sorts, tree constructor operations to constructor operations, etc. Most constructor operations are free, except for all operations corresponding to List or Sequence concatenation, see & in (2.2-1). In the case of STMT_SEQ, Empty corresponds to **null**; in Ada and & would correspond to ; in the concrete syntax for Pascal-like languages; cf. [11].

(2.1-1) Example: Algebra of Data: Monoid (INT, *, 1)

axiom ∀ X,Y,Z : INT ⇒
(X * Y) * Z = X * (Y * Z), 1 * X = X, X * 1 = X

(2.1-2) Example: Algebra of Programs: Monoid (STMT_SEQ, &, Empty)

axiom ∀ R, S, T : STMT_SEQ ⇒
(R & S) & T = R & (S & T), Empty & R = R, R & Empty = R

2.2. Concrete and Abstract Syntax

Although we are interested in the algebra of programs, that is the abstract syntax, it is often more convenient to use a notation enclosed in ⌜ ⌟ for *phrases* (program fragments with schema variables) of the concrete syntax corresponding to appropriate *terms* (with variables) in the algebra of programs, see (2.2-1). Variables correspond in the example, n-fold repetition of (possibly distinct) actual parameter expressions *E1* in ⌜ {*E1*, }^{n} ⌟ corresponds to the list EL1 of expressions (simplified parameter associations) of length n , etc. In this paper, we are not concerned with notational issues at the concrete syntax level nor with the (non-trivial) translation of phrases from concrete to abstract syntax. Also, the typing and universal quantification of variables in equational axioms is usually omitted for brevity; it should be apparent from the context.

(2.2-1) Example: Correspondence between Concrete and Abstract Syntax: Procedure Call

⌜ *P*({*E1*, }^{n} *V* {, *E2*}); ⌟ ~ Call(*P*, *EL1* & Exp(*V*) & *EL2*))

2.3. Algebraic Semantics

In the approach of the algebraic definition of the semantics of a programming language (cf. [12]), an evaluation function or interpretation function from syntactic to semantic domains is axiomatised. The equational axioms of such functions induce equivalence classes on (otherwise free) constructor terms. In other words, we can prove that two (syntactic) terms are *semantically equivalent*, in a context-free way or possibly subject to some syntactic or semantic pre-conditions. Such a proof can of course also be made with respect to some other style of semantic definition for the language. Thus we obtain a *semantic algebra* of programs in which transformation rules are identities as a quotient algebra of the *abstract syntatic algebra* in which only identities for & exist.

Note that the semantic specification may be intentionally loose, that is some semantic aspects such as the order of evaluation of expressions in a call may be intentionally left unspecified. From an algebraic point of view this means that several distinct semantic models exist for the loose semantic specification. Usually, these form a lattice between the initial model on top (where all terms are distinct that cannot be proven to equal) and the terminal model at the bottom (where all terms are the same that cannot be proven to differ). In some cases, unique initial and terminal models may not exist: if expressions may have side-effects, for example, several (quasi-terminal) models exist according to particular sequentialisations of evaluation (see below). Each choice of model (each choice of sequentialisation by a compiler) is admissable. (In Ada, a program is erroneous, if the quasi-terminal semantic models for this program do not coincide.)

3. Transformation Rules

3.1. Examples of Transformation Rules

Consider the examples below: (3.1-1) is actually a specialisation of a more general rule for the (arbitrary) Introduction ⇔ Elimination of a Declaration. Rule (3.1-2) transforms function calls of a particular kind into procedure calls. Rule (3.1-3) brings an assignment of a function call to a variable into the form required by rule (3.1-2). Finally, rule (3.1-4) unnests expressions that might contain a call to F such that rule (3.1-3) can be applied. We are assuming a (sub)language (of Ada) where expression have no side-effects (see also section 3.5).

Note that, in a proper formulation for these rules, the context has to be taken into account, to ensure that (3.1-2) is applied in the context of declarations for F and P as introduced by (3.1-1), that V or T are properly declared, and that T (or V in (3.1-4)) do not occur in the right hand context unless previously assigned to. For lack of space, we are ignoring these considerations here; see [13] for a particular approach to a specification of context in an algebraic framework.

Assuming suitable context conditions (in particular for (3.1-2)), each rule is applicable by itself and correct as an individual rule. In a combined transformation (or "development"), all these rules are taken together (with suitable unification of F, n etc., and some additional context conditions omitted for brevity) and applied exhaustively in reverse order; they will then transform the function F and all its calls to the procedure P and F can be eliminated. Before we come to the issue of application strategies etc. in chapter 6, let us consider some rules in more detail.

(3.1-1) Trafo Rule: Introduction ⇔ Elimination of Procedure Declaration

```
declare
  {D1}
  function F({FP1; }^n X: R  {; FP2}) return R;

  {D2}
begin
  {S}
end;
such that
• P does not occur in any of the D2, S
• P is not in conflict with other declarations
```

```
declare
  {D1}
  function F({FP1; }^n X: R  {; FP2}) return R;
  procedure P({FP1; }^n X: in out R  {; FP2});
  {D2}
begin
  {S}
end;
```

(3.1-2) Trafo Rule: Assignment with Function Call ⇔ Procedure Call

```
V := F({E1, }^n V {, E2});
```

```
P({E1, }^n V {, E2});
```

(3.1-3) Trafo Rule: Collateral ⇔ Sequential Evaluation of Expressions in Function Call

```
V := F ({E1, }^n E {, E2});
such that
• V does not occur in any of the E1, E2
```

```
V := E;
V := F ({E1, }^n V {, E2});
```

```
V := F ({E1, }^n E {, E2});
such that
• V does occur in one of the E1, E2
• T does not occur in any of the E1, E, E2
```

```
T := V; V := E;
V := F ({E1[V by T], }^n V {, E2[V by T]});
```

(3.1-4) Trafo Rule: Nested ⇔ Sequential Evaluation of Expressions in Statements

```
( W := G({E1, } E {, E2});
| Q({E1, } E {, E2});
| if E then {S3} [else {S4} ] end if;
| while E loop {S3} end loop; )
such that
• V does not occur in any of the E1, E, E2, S3, S4
```

```
  V := E;
( W := G({E1, } V {, E2});
| Q({E1, } V {, E2});
| if V then {S3} [else {S4} ] end if;
| while V loop {S3} V := E; end loop; )
```

3.2. Bi-Directional Rules: Equations

The first and major kind of transformation rules we are interested in is the *bi-directional transformation rule*, a pair of semantically equivalent terms (in the above sense), that is an equation

in the algebra of programs that is provable by deductive or inductive reasoning against the semantics. All rules in this paper are of this kind (indicated by ⇔); but see also section 3.5.

(3.2-1) is just a translation of the concrete syntax in (3.1-2) to terms in the algebra of programs, cf. also (2.1-3); a similar translation for (3.1-1) would look very involved. (3.2-2) is an auxiliary rule; applicability conditions have been formalised using auxiliary functions (see section 3.6) and make the equation conditional. The condition ¬ *IsVar(* E *)* is not strictly necessary (as in most other cases below) but restricts the application of the rule such that no trivial assignments (corresponding to renamings of variables) are produced. (3.2-3) is a translation of (3.1-3). As stated above, the context has to be taken into account to ensure that ¬ *OccursIn(* V, S *)* etc. extend properly over the right hand context.

(3.2-1) Trafo Rule: Assignment with Function Call ⇔ Procedure Call: as Equation

AssignStmt(*V*, Call(*F*, *EL1* & Exp(*V*) & *EL2*)) = Call(*P*, *EL1* & Exp(*V*) & *EL2*))

(3.2-2) Trafo Rule: Multiple ⇔ Single Evaluation of Same Subexpression

OccursIn(E, S *)* ∧ ¬ *OccursIn(* V, S *)* ∧ ¬ *IsVar(* E *)* →
S = ⌈ V := E; *SubstByIn(* E, V, S *)* ⌋

(3.2-3) Trafo Rule: Collateral ⇔ Sequential Evaluation in Assignment with Function Call

¬ *OccursIn(* V, EL1 *)* ∧ ¬ *OccursIn(* V, EL2 *)* ∧ ¬ *IsVar(* E *)* →
⌈ V := F (EL1, E, EL2); ⌋ = ⌈ V := E; V := F (EL1, V, EL2); ⌋,
OccursIn(V, ⌈ EL1, EL2 ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ EL1, E, EL2 ⌋ *)* ∧ ¬ *IsVar(* E *)* →
⌈ V := F (EL1, E, EL2); ⌋ =
⌈ T := V; V := E; V := F (*SubstByIn(* V, T, EL1 *)*, V, *SubstByIn(* V, T, EL2 *)*); ⌋

3.3. Derivation of Transformation Rules

The first equation in (3.2-3) is generally applicable (cf. (3.1-4)), but is not sufficient for our goal, namely that the call should have the particular form, even if V does occur in the other subexpressions. Let us now derive the second equation of (3.2-3) from the first, using the generally applicable rule (3.2-2) as a start, see (3.3-1). We apply the usual derivation steps of substitution, application of an equational law, renaming of variables, e.g. V by T in the first equation of (3.2-3) before it is applied as a law.

(3.3-1) Trafo Rule Derivation: Collateral ⇔ Sequential Evaluation for Function Call

OccursIn(V, E *)* ∧ ¬ *OccursIn(* T, E *)* →
⌈ V := E; ⌋ = ⌈ T := V; V := *SubstByIn(* V, T, E *)* ; ⌋

OccursIn(V, ⌈ F (EL1, E, EL2) ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ F (EL1, E, EL2) ⌋ *)* →
⌈ V := F (EL1, E, EL2); ⌋ =
⌈ T := V; V := *SubstByIn(* V, T, ⌈ F (EL1, E, EL2) ⌋ *)* ; ⌋

OccursIn(V, ⌈ EL1, E, EL2 ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ EL1, E, EL2 ⌋ *)* →
⌈ V := F (EL1, E, EL2); ⌋ =
⌈ T := V; V := F (*SubstByIn(* V, T, ⌈ EL1, E, EL2 ⌋ *)*); ⌋

OccursIn(V, ⌈ EL1, EL2 ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ EL1, E, EL2 ⌋ *)* →
⌈ V := F (EL1, E, EL2); ⌋ =
⌈ T := V; V := F (*SubstByIn(* V, T, EL1 *)*, E, *SubstByIn(* V, T, EL2 *)*); ⌋

OccursIn(V, ⌈ EL1, EL2 ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ EL1, E, EL2 ⌋ *)* →
⌈ V := F (EL1, E, EL2); ⌋ =
⌈ T := V; V := E; V := F (*SubstByIn(* V, T, EL1 *)*, V, *SubstByIn(* V, T, EL2 *)*); ⌋

3.4. Sets of Transformation Rules

We may have already noticed in section 3.1 that each rule in a set of rules achieves a certain (sub)goal (possibly only when applied exhaustively) that makes another rule applicable. We will come back to this issue below. For the time being let us consider different sets of rules that achieve the same effect.

(3.4-1) is an analogous rule to (3.2-3) and arises, for example, during recursion removal when a tail recursive call is transformed to a collateral assignment of actual parameter expressions to variables corresponding to formal parameters. Collateral (or multiple) assignments do not exist in Ada, but we could define them as intermediate notational extensions, to be eliminated into sequences of individual assignments during the course of program development. We assume that V := V; (an identity assignment) can be replaced by a null-statement even if V is not initialised.

The first equation in (3.4-1) terminates the recursion by removing (a list of) redundant identity assignments, the latter two are derived from (3.4-1).

(3.4-2) is a similar rule where the removal of identity assignments is reflected in the contraction of the lists. In fact the second equation is a specialised case of the third for E equals V and removal of a redundant identity assignment; such a specialisation, and appropriate guards, could also be introduced in (3.4-1).

(3.4-3) is a simple-minded specialisation that forces a sequentialisation from left to right. This way, it does not avoid redundant assignments and auxiliary ("temporary") variables.

The important observation is that all these (sets of) rules, when applied exhaustively, yield semantically equivalent sequentialisations of the collateral assignment. With respect to some efficiency metrics where minimisation of assignments and variable usage is a concern, however, they behave quite differently. Moreover, the order of application of a general rule (rather than simple-minded application from left-to-right) becomes of great importance. Each application strategy yields a different syntactic (normal) form.

(3.4-1) Trafo Rule: Collateral ⇔ Sequential Assignment

⌈ (NL) := (NL); ⌋ = ⌈ **null**; ⌋

Length(NL1 *)* = *Length(* EL1 *)* ∧ ¬ *OccursIn(* V, EL1 *)* ∧ ¬ *OccursIn(* V, EL2 *)* →
⌈ (NL1, V, NL2) := (EL1, E, EL2); ⌋ =
⌈ V := E; (NL1, V, NL2) := (EL1, V, EL2);⌋

Length(NL1 *)* = *Length(* EL1 *)* ∧
OccursIn(V, ⌈ EL1, EL2 ⌋ *)* ∧ ¬ *OccursIn(* T, ⌈ EL1, E, EL2 ⌋ *)* →
⌈ (NL1, V, NL2) := (EL1, E, EL2); ⌋ =
⌈ T := V; V := E; (NL1, V, NL2) := (*SubstByIn(* V, T, EL1 *)*, V, *SubstByIn(* V, T, EL2 *)*);⌋

(3.4-2) Trafo Rule: Collateral ⇔ Sequential Assignment

⌈ (V) := (E); ⌋ = ⌈ V := E; ⌋

Length(NL1 *)* = *Length(* EL1 *)* ∧ *Length(* NL1 *)* + *Length(* NL2 *)* > 0 →
⌈ (NL1, V, NL2) := (EL1, V, EL2); ⌋ = ⌈ (NL1, NL2) := (EL1, EL2); ⌋

$$Length(\,NL1\,) = Length(\,EL1\,) \wedge Length(\,NL1\,) + Length(\,NL2\,) > 0 \wedge$$
$$\neg\, OccursIn(\,V, EL1\,) \wedge \neg\, OccursIn(\,V, EL2\,) \rightarrow$$
$$\lceil\, (\,NL1, V, NL2\,) := (\,EL1, E, EL2\,);\rfloor = \lceil\, V := E; (\,NL1, NL2\,) := (\,EL1, EL2\,);\rfloor$$

$$Length(\,NL1\,) = Length(\,EL1\,) \wedge Length(\,NL1\,) + Length(\,NL2\,) > 0 \wedge$$
$$OccursIn(\,V, \lceil\, EL1, EL2\,\rfloor) \wedge \neg\, OccursIn(\,T, \lceil\, EL1, E, EL2\,\rfloor) \rightarrow$$
$$\lceil\, (\,NL1, V, NL2\,) := (\,EL1, E, EL2\,);\rfloor =$$
$$\lceil\, T := V; V := E; (\,NL1, NL2\,) := (\,SubstByIn(\,V, T, EL1\,), SubstByIn(\,V, T, EL2\,)\,);\rfloor$$

(3.4-3) Trafo Rule: Collateral ⇔ Sequential Assignment (Specialisation: Left to Right)

$$\lceil\, (\,V\,) := (\,E\,);\rfloor = \lceil\, V := E;\rfloor$$

$$Length(\,NL\,) > 0 \wedge \neg\, OccursIn(\,T, EL\,) \rightarrow$$
$$\lceil\, (\,V, NL\,) := (\,E, EL\,);\rfloor = \lceil\, T := V; V := E; (\,NL\,) := (\,SubstByIn(\,V, T, EL\,)\,);\rfloor$$

3.5. Uni-Directional Rules: Relations

If the evaluation of an expression may have side-effects, that is for a semantics with distinct quasi-terminal models (see section 2.2 above), the rules (3.2-2) to (3.4-3) could not be bi-directional any more. The equality would have to be replaced by a semantic inclusion relation in a model-oriented sense. Thus a *uni-directional* transformation rule is a relation between semantic models such that each model in the range is a robustly correct implementation of some model in the domain. Again this notion is taken from the theory of algebraic specification (cf. [10]) and formalises the intuitive notion of correctness with respect to some implementation decision that narrows implementation flexibility or chooses a particular one. These rules are of course not invertible (a decision cannot be reversed) and, interpreted as rewrite rules, are not confluent in general. In this paper, we cannot go into detail and restrict our attention to bi-directional rules although most considerations generalise.

4. Transformation Operations

4.1. Auxiliary Operations

We note a number of auxiliary functions and predicates in the above equations, such as *SubstByIn* or *OccursIn*. They can be structurally defined as in (4.1-1) and must hold over subterms or over a larger context of the actual rule application, see chapter 6 below. Such functions could be represented by (derived or inherited) attributes in an implementation of transformation rules by attributed tree transformations.

(4.1-1) Auxiliary Operation: Substitution in Expressions

$$\neg\, OccursIn(\,E1, E2\,) \rightarrow SubstByIn(\,E1, T, E2\,) = E2\,,$$
$$SubstByIn(\,E, T, E\,) = T,$$
$$OccursIn(\,E, EL\,) \rightarrow SubstByIn(\,E, T, \lceil\, F(\,EL\,)\,\rfloor) = \lceil\, F(\,SubstByIn(\,E, T, EL\,)\,)\,\rfloor$$

SubstByIn and *OccursIn* would be similarly defined for other kinds of terms, for example Expression Lists (see chapter 5 below).

4.2. Transformation Operations: Endomorphisms

An elementary transformation operation can be constructed from (transformation rule(s), that is) equation(s) in the semantic algebra in a straightforward way as a partial function in the abstract syntactic algebra, see (4.2-1): it maps to a normal form in the quotient algebra obtained by dividing by the equation(s). It embodies the effect of considering each equation as a rewrite rule from left to right or from right to left, depending on the chosen normal form. Thus it achieves a normalisation in the abstract syntactic algebra and corresponds to an identity in the semantic algebra.

(4.2-1) Trafo Operation: Assignment with Function Call to Procedure Call

TrafoCall(AssignStmt(*V*, Call(*F*, *EL1* & Exp(*V*) & *EL2*))) = Call(*P*, *EL1* & Exp(*V*) & *EL2*)

4.3. Extension of the Domain

If we want to apply elementary transformations over a larger context, with some strategy such as somewhere or everywhere (see chapter 6), we need to extend the domain of a partial function to larger terms, as in (4.3-1) for TrafoCall. The first equation corresponds to the previous definition for TrafoCall in a slightly different formulation. The second and third extend the definition to the identity over STMT in all other cases.

(4.3-1) Trafo Operation: Extension to STMT (Monomorphic Specification)

EqStmt(S, AssignStmt(*V*, Call(*F*, *EL1* & Exp(*W*) & *EL2*))) ∧ EqName(*V*, *W*) →
  TrafoCall(S) = Call(*P*, *EL1* & Exp(*V*) & *EL2*),
EqStmt(S, AssignStmt(*V*, Call(*F*, *EL1* & Exp(*W*) & *EL2*))) ∧ ¬ EqName(*V*, *W*) →
  TrafoCall(S) = S,
¬ EqStmt(S, AssignStmt(*V*, Call(*F*, *EL1* & Exp(*W*) & *EL2*))) →
  TrafoCall(S) = S

5. Development of Transformation Operations

5.1. Loose Specification

(5.1-1) Trafo Operation: Extension to STMT (Polymorphic Specification)

(TrafoCall(AssignStmt(*V*, Call(*F*, Exp(*V*) & *L*))) = Call(*P*, Exp(*V*) & *L*))) ∨ (TrafoCall(S) = S)

Compared with (4.3-1), the compact definition of (5.1-1) is also semantically correct since TrafoCall is an endomorphism and therefore all values in the equivalence class denoted by the original rule are acceptable. Loose specifications allow several distinct (that is non-isomorphic) models. In this case the ∨ operator between equations has been used to allow an additional degree of freedom over classical horn-clause specifications, analogous to non-determinacy. This version specifies a class of functions (one being the "syntactic" identity in the term algebra); the more explicit definition of (4.3-1) specifies a single function mapping to a canonical form: for each non-trivial application the function call is actually changed to a procedure call.

Such a simple definition is often convenient at the start of the development of a transformation operation to characterise its effect before turning to considerations of termination, efficiency etc.

5.2. Requirement and Design Specifications

In general, we would like to start with a *requirement specification* of a transformation operation before considering a particular *design specification*, possibly several design alternatives (cf. also section 3.4). The same kind of reasoning as in program development can be applied. Any of the designs is then either formally derived from or proved to be a (robustly) correct implementation of the requirement specification (cf. [9,10]).

As an example, consider the extension of TrafoCall the effect of over STMT_SEQ. We can characterise the desired effect as in (5.2-1): TrafoCall should be applied to every element of a sequence (alternatively: to some arbitrary element). (5.2-2) and (5.2-3) show two divide and conquer strategies for achieving this (cf.[14]), depending on the basic operations available on STMT_SEQ, a partition or left linear structural decomposition strategy is applied. In fact, we can abbreviate such strategies by functional abstraction using a functional as in (5.2-4), see section 6.2.

(5.2-1) Trafo Operation: Extension over STMT_SEQ: *Requirement Specification*

TrafoCallStmts (Empty) = Empty,
I > 0 ∧ I < Length(SSeq) → Select (TrafoCallStmts(SSeq), I) = TrafoCall (Select(SSeq, I))

(5.2-2) Trafo Operation: Extension over STMT_SEQ: *Design Specification:* Partition

TrafoCallStmts(Empty) = Empty, TrafoCallStmts(SSeq1 & S & SSeq2) = TrafoCallStmts(SSeq1) & TrafoCall(S) & TrafoCallStmts(SSeq2)

(5.2-3) Trafo Operation: Extension over STMT_SEQ: *Design Specification:* Linear Decompos.

TrafoCallStmts (Empty) = Empty, TrafoCallStmts (Add(S,R)) = Add (TrafoCall(S), TrafoCallStmts(R))

(5.2-4) Trafo Operation: Extension over STMT_SEQ: *Design Specification:* Functional

TrafoCallStmts(SSeq) = MapStmtSeq (TrafoCall)(SSeq)

6. Functionals

6.1. Homomorphic Extension

Before we continue with development considerations, let us focus on this issue of functional abstraction in more detail. Higher order functions allow a substantial reduction of re-development effort (just as parameterised data type specifications), just as in program development (cf. [15]).

In fact, most of these functionals have the nature of *homomorphic extension* functionals (see [16]), in this case the structural extension of the effect of a (local) transformation or predicate over larger terms. Compare the definition of auxiliary operations and predicates using functionals in (6.1-1). *EveryWhere* and *EveryWherePred* would be similarly defined for other kinds of terms. *EveryWhere* is an endomorphic extension functional from terms to terms, and *EveryWherePred* is a homomorphic extension functional from terms to BOOLEAN.

(6.1-1) Operation Functional: Extension over Expression Lists

SubstByIn(T1, T, T2 *)* = *EveryWhere(SubstByInE(* T1, T*))* *(* T2 *)*
¬ *OccursIn(* E1, E2 *)* → *SubstByInE(* E1, T*)* *(* E2 *)* = E2 , *SubstByInE(* E, T *)* *(* E*)* = T, *OccursIn(* E, EL *)* → *SubstByInE(* E, T*)* *(*⌈ F(EL) ⌋ = ⌈ F(*SubstByIn(* E, T, EL *)*) ⌋
IsExp(E *)* → *EveryWhere(* F*)* *(* E *)* = F *(* E *)* , *EveryWhere(* F*)* *(*⌈ EL1, EL2 ⌋ = ⌈ *EveryWhere(* F*)* *(* EL1 *)*, *EveryWhere(* F*)* *(* EL2 *)* ⌋

OccursIn(T1, T2 *)* = *EveryWherePred(OccursInE(* T1*))* *(* T2 *)*
OccursInE(E *)* *(* E *)* = *TRUE* , ¬ *EqName(* N1, N2 *)* → *OccursInE(* N1*)* *(* N2 *)* = *FALSE* , *OccursInE(* E *)* *(*⌈ F(EL) ⌋ = *OccursIn(* E, EL *)*
IsExp(E *)* → *EveryWherePred(* Pred*)* *(* E*)* = Pred *(* E *)* , *EveryWherePred(* Pred*)* *(*⌈ EL1, EL2 ⌋ = *EveryWherePred(* Pred*)* *(* EL1 *)* ∧ *EveryWherePred(* Pred*)* *(* EL2 *)*

6.2. Restricted Functionals

Similarly, we can abstract the homomorphic extension over statement sequences in (5.1-1) to (5.1-3) to a functional. (6.2-1) shows the signature, an abstract requirement specification and a particular design specification by partition.

It is an interesting observation that most definitions of such functionals have a restricted form: the functional argument is unchanged in recursive calls. Functionals of this restricted form can be transformed to Ada generics since they can be interpreted as more or less textual abbreviations; instantiation is then explicit, see (6.2-2). A functional together with its functional parameters can then always be considered as a new function symbol (corresponding to an implicit instantiation),

therefore the conformance to the theory of algebraic specification is evident in the restricted case. In this paper, we will restrict ourselves to this case. In the presence of overloading, a functional that is locally defined to a parameterised specification has the same effect as a polymorphic functional.

(6.2-1) Operation Functional: Extension over STMT_SEQ

```
function MapStmtSeq ( G: function (S: STMT) return STMT )
                    return function (SSeq: STMT_SEQ) return STMT_SEQ;

axiom for all G: function (S: STMT) return STMT; SSeq, SSeq1, SSeq2: STMT_SEQ; I: NATURAL ⇒
   MapStmtSeq (G)(Empty) = Empty,
   I > 0 ∧ I < Length(SSeq) → Select ( MapStmtSeq (G)(SSeq), I) = G (Select(SSeq, I));

   MapStmtSeq(G)(Empty) = Empty,
   MapStmtSeq(G)(SSeq1 & S & SSeq2) = MapStmtSeq(G)(SSeq1) & G(S) & MapStmtSeq(G)(SSeq2)
```

(6.2-2) Operation Functional: Extension over STMT_SEQ: as Ada Generic

```
generic
   with function G (S: STMT) return STMT;
function  MapStmtSeq (SSeq: STMT_SEQ) return STMT_SEQ;
   axiom for all SSeq, SSeq1, SSeq2: STMT_SEQ; I: NATURAL ⇒
      MapStmtSeq (Empty) = Empty,
      I > 0 ∧ I < Length(SSeq) → Select ( MapStmtSeq (SSeq), I) = G (Select(SSeq, I));

      MapStmtSeq(Empty) = Empty,
      MapStmtSeq(SSeq1 & S & SSeq2) = MapStmtSeq(SSeq1) & G(S) & MapStmtSeq(SSeq2)

• Instantiation:
function TrafoCallStmts (SSeq: STMT_SEQ) return STMT_SEQ is new MapStmtSeq (TrafoCall);
   ... TrafoCallStmts (SSeq) ...
```

6.3. Transformals

In analogy to tacticals in [17], we might call transformation functionals *transformals* since they embody application tactics or strategies. Compare the differences in the definition of the homomorphic extension functionals *SomeWhere* and *EveryWhere; F* must be a total function over a sub-domain, for example on simple statements. Note that the ∨ operator between equations has been used (cf. section 5.1) to indicate arbitrary choice between then or else part, for example. Thus the function denoting a particular occurrence of an application of F is in the specified class of functions. *IterateWhile* can be used to apply a transformation function *F* as long as some condition *C* holds. Similarly, *IterateSomeWhile* iterates a local transformation function *F* as long as some local condition *C* holds somewhere.

(6.3-1) Operation Functional: SomeWhere

```
IsSimpleStmt(Stmt) → SomeWhere (F) (Stmt) = F (Stmt),
( SomeWhere (F)(   ⌈ SSeq1 SSeq2 ⌋) = ⌈ SomeWhere (F) (SSeq1) SSeq2 ⌋ ) ∨
  (SomeWhere (F)(  ⌈ SSeq1 SSeq2 ⌋) = ⌈ SSeq1 SomeWhere (F) (SSeq2) ⌋ ),
( SomeWhere (F)(   ⌈ if B then SSeq1 [ else SSeq2 ] end if; ⌋ ) =
                   ⌈ if B then SomeWhere (F) (SSeq1) [ else SSeq2 ] end if; ⌋ ) ∨
  ( SomeWhere (F)( ⌈ if B then SSeq1 else SSeq2 end if; ⌋ ) =
                   ⌈ if B then SSeq1 else SomeWhere (F) (SSeq2) end if; ⌋ ),
SomeWhere (F)(   ⌈ while B loop SSeq end loop; ⌋ ) =
                 ⌈ while B loop SomeWhere (F) (SSeq) end loop; ⌋
```

(6.3-2) Operation Functional: EveryWhere

```
IsSimpleStmt(Stmt) → EveryWhere (F) (Stmt ) = F (Stmt),
EveryWhere (F)( ⌈ SSeq1 SSeq2 ⌋) = ⌈ EveryWhere (F) (SSeq1)  EveryWhere (F) (SSeq2) ⌋
EveryWhere (F)( ⌈ if B then SSeq1 [ else SSeq2 ] end if; ⌋) =
                ⌈ if B then EveryWhere (F) (SSeq1) [ else EveryWhere (F) (SSeq2) ] end if; ⌋,
EveryWhere (F)( ⌈ while B loop SSeq end loop; ⌋) =
                ⌈ while B loop EveryWhere (F) (SSeq) end loop; ⌋
```

(6.3-3) Operation Functional: Iterate

```
¬ C(X) →  IterateWhile (F, C) (X) = X,
  C(X) →  IterateWhile (F, C) (X) = IterateWhile (F, C) (F(X)),
IterateSomeWhile (F, C) = IterateWhile (SomeWhere(F), SomeWherePred(C))
```

7. Developments

7.1. Developments: Composite Transformations

Since we can regard every elementary program development step as a transformation, we may conversely define a *development* to be a composition of transformations or term over transformation operations (including application strategies for elementary transformation operations). This view is independent of whether we regard a development to formalise a concrete development history (a term without variables), possibly to be replayed, or a development method (a term with variables or a functional abstraction) for future application.

7.2. Development Goals

We have already stated in chapter 3 that the application of some (set of) rule(s) often requires the satisfaction of some pre-condition established by (exhaustive application of) some other (set of) rule(s). Conversely, this condition can be considered to be a required post-condition of the first (set of) rule(s), or a characteristic predicate for the respective transformation function. Let us call such a condition a *development goal*: it is a requirement specification for a function yet to be designed.

If these conditions can be defined structurally (or "syntactically"), as we indeed hope will mostly be the case, then they characterise certain *normal forms*. This leads to a substantial improvement in the modularisation of sets of rules and separation of concerns, consequently ease of verification. Note that intermediate conditions never need to be checked operationally as long as it can be proved that they are established by previous application of other rules.

(7.2-1) shows the pre-conditions for rules (3.1-3) and (3.1-2) (or the corresponding rules in section 3.4), resp., and the precondition for finally eliminating the function declaration altogether

(7.2-1) Development Goals: Function to Procedure

```
NoNestedCall(F)(S) =          EveryWherePred(NotIsNestedCall(F))(S),
EveryCallUpdateForm(F, n)(S) = EveryWherePred(IsCallUpdateForm(F, n))(S),
NoCall(F)(S) =                EveryWherePred(NotIsCall(F))(S)
```

7.3. Composition of Developments

We can now formulate the definition for the individual transformation functions achieving sub-goals of the development, and the overall function *TFunctToProc* as a functional composition.

(7.3-1) Development: Function to Procedure

```
ProperContext(F, n)(Scope) →
  TUnnestEveryCall(F)(Scope) = IterateSomeWhile (TUnnestCall(F), IsCall(F))(Scope),
NoNestedCall(F)(Scope) →
  TEveryCallUpdateForm(F, n)(Scope) =
    IterateSomeWhile (TCallUpdateForm(F, n), NotIsCallUpdateForm(F, n))(Scope),
EveryCallUpdateForm(F, n)(Scope) →
  TEveryCallToProc(F, n, P)(Scope) = IterateSomeWhile (TCallToProc(F, n, P), IsCall(F))(Scope),
NoCall(F)(Scope) →
  TElimFunctDecl(F)(Scope) = SomeWhere (TElimDecl(F))(Scope)
```

```
ProperContext(F, n)(Scope) →
  TFunctToProc(F, n, P)(Scope) =
    TElimFunctDecl(F)
      TEveryCallToProc(F, n, P) TEveryCallUpdateForm(F, n) TUnnestEveryCall(F)
        IntroProcDecl(F, n, P) (Scope)
```

7.4. Development Rules

We note that it makes no difference in (7.3-1) whether to introduce the procedure declaration before or after normalisation of the function calls. This can been expressed by a re-ordering property as in (7.4-1). Such properties or *development rules* allow us to express and to reason about design alternatives or *alternative development strategies* and to*simplify developments* by considering them as algebraic terms in the usual way, cf. (7.4-2).

Such reasoning leads to considerable simplification of the above development. For example, one would want to simplify iterated application into bottom-up one-sweep (everywhere) application, or a sequence of exhaustive individual application of one rule, then the other, to exhaustive application of both rules combined, whenever possible. Due to lack of space, such simplifications cannot be included here. The important observation is that the theoretical reasoning and practical technique for the development of transformation functions (or "developments") here is the same as the established one for programs.

(7.4-1) Development Rule: Reordering Property of Transformations

```
TEveryCallUpdateForm(F, n) TUnnestEveryCall(F) IntroProcDecl(F, n, P) (Scope) =
  IntroProcDecl(F, n, P) TEveryCallUpdateForm(F, n) TUnnestEveryCall(F) (Scope)
```

(7.4-2) Development Rule: Elimination of Iteration

```
C(X) ∧¬ C(F(X)) →  IterateWhile (F, C) (X) = F(X)
```

8. Conclusion

It has been demonstrated that the methodology for program development based on the concept of algebraic specification of data types and program transformation can be applied to the development of transformation operations ("transformation programs"); in the algebra of programs, equations correspond to bi-directional transformation rules. Starting from small elementary transformation rules that are proved correct against the semantics of the programming language, we can apply the usual equational and inductive reasoning to develop complex rules; we can reason about development goals as requirement specifications for transformation operations and characterise them as structural normal forms; we can implement them by various design strategies; we can optimise them using algebraic properties; we can use composition and functional abstraction; in short, we can develop *correct*, efficient, complex transformation operations from elementary algebraic properties.

Moreover, we can regard program development ("histories") as formal objetcs: as (compositions of) such transformation operations. We can specify development goals, implement them using available operations, simplify development terms, re-play developments by interpretation, and abstract from concrete developments to development methods, that is formalised development tactics and strategies. There is a close analogy to the development of efficient proof strategies for given

inference rules (transformation rules in the algebra of proofs). Perhaps the approach could also be used to formalise rule and inference based expert systems.

Since every manipulation in a program development system can be regarded as a transformation of some "program" (for example in the command language), the whole system interaction can be formalised this way and the approach leads to a uniform treatment of programming language, program manipulation and transformation language, and command language.

Acknowledgements

I wish to thank M. Broy, H. Ganzinger, B. Gersdorf, S. Kahrs, D. Plump, and Z. Qian for helpful criticism and comments.

References

[1] Wile, D. S..: Program Developments: Formal Explanations of Implementations. *Comm. ACM 26:* 11(1983) 902-911. *also in:* Agresti, W. A. (ed.): *New Paradigms for Software Development.* IEEE Computer Society Press / North Holland (1986) 239-248.

[2] Steinbrüggen, R..: Program Development using Transformational Expressions. Rep. TUM-I8206, Institut für Informatik, TU München, 1982.

[3] Feijs, L.M.G., Jonkers, H.B.M, Obbink, J.H., Koymans, C.P.J., Renardel de Lavalette, G.R., Rodenburg, P.M.: A Survey of the Design Language Cold. *in:* Proc ESPRIT Conf. 86 (Results and Achievements). North Holland (1987) 631-644.

[4] Jähnichen, S., Hussain, F.A., Weber, M.: Program Development Using a Design Calculus. *in:* Proc ESPRIT Conf. 86 (Results and Achievements). North Holland (1987) 645-658.

[5] Bauer, F.L., Berghammer, R., Broy, M., Dosch, W., Geiselbrechtinger, F., Gnatz, R., Hangel, E., Hesse, W., Krieg-Brückner, B., Laut, A., Matzner, T., Möller, B., Nickl, F., Partsch, H., Pepper, P., Samelson, K., Wirsing, M., Wössner, H.: The Munich Project CIP, Vol. 1: The Wide Spectrum Language CIP-L. *LNCS 183,* 1985.

[6] Bauer, F. L., Wössner, H.: *Algorithmic Language and Program Development.* Springer 1982.

[7] Pepper, P.: A Simple Calculus of Program Transformations (inclusive of Induction). Rep. TUM-I8409, Institut für Informatik, TU München, 1984.

[8] Krieg-Brückner, B., Hoffmann, B., Ganzinger, H., Broy, M., Wilhelm, R., Möncke, U., Weisgerber, B., McGettrick, AA.D., Campbell, I.G., Winterstein, G.: Program Development by Specification and Transformation. *in:* Proc ESPRIT Conf. 86 (Results and Achievements). North Holland (1987) 301-312.

[9] Krieg.Brückner, B.: Integration of Program Construction and Verification: the PROSPECTRA Project. in: Habermann, N., Montanari, U. (eds.): Innovative Software Factories and Ada. Proc. CRAI Int'l Spring Conf. '86. *LNCS* (to appear).

[10] Broy, M., Wirsing, M.: Partial Abstract Types. *Acta Informatica 18* (1982) 47-64.

[11] Hoare, C.A.R.: Mathematics of Programing. *BYTE* (1986) 115-149.

[12] Broy, M., Pepper, P., Wirsing, M.: On the Algebraic Definition of Programming Languages. *ACM TOPLAS 9* (1987) 54-99.

[13] Qian, Z.: Structured Contextual Rewriting. Proc. Int'l Conf. on Rewriting Techniques and Applications (Bordeaux). *LNCS 256* (1987) 168-179.

[14] Smith, D.R.: Top-Down Synthesis of Divide-and-Conquer Algorithms. *Artifical Intelligence 27:*1 (1985) 43-95.

[15] Bird, R.S.: Transformational Programming and the Paragraph Problem. *Science of Computer Programming 6* (1986) 159-189.

[16] von Henke, F.W.: An Algebraic Approach to Data Types, Program Verification and Program Synthesis. *in:* Mazurkiewicz, A. (ed.): Mathematical Foundations of Computer Science 1976. *LNCS 45* (1976) 330-336.

[17] Gordon, M., Milner, R., Wadsworth, Ch.: Edinburgh LCF: A Mechanised Logic of Computation. *LNCS 78* .

Project No. 125

TERM REWRITING SYSTEMS IN THE GRASPIN ENVIRONMENT USED FOR THE VERIFICATION OF SOFTWARE DEVELOPMENT STEPS

B. DEHM[1],H-R. FONIO[2],H.GERLACH[3],W. SOMMER[3],R.TOBIASCH[1]

[1] Siemens AG,Zentrale Forschung und Entwicklung,
Otto-Hahn-Ring 6, D-8000 München 83

[2] Gesellschaft für Mathematik und Datenverarbeitung,
Institut für Systemtechnik, Postfach 1240, D-5205 Sankt Augustin 1

[3] Universität Kaiserslautern,Fachbereich Informatik
Postfach 3049, D-6750 Kaiserslautern

Abstract

The GRASPIN validation and verification scenario was designed in order to make existing techniques and methods applicable to the working software engineer.We present two methods MISOP (Method for the Check of Important Specification Object Properties) and MCAI (Method for the check of the Correctness of an Algebraic Implementation) using term rewriting techniques for the verification of software development steps. Verification means here to ensure type-protection, operation-completeness, and RI-correctness as defined by Ehrig et al.. A modified Knuth-Bendix-Algorithm is used for confluence and consistency checks. For proving termination of term rewriting systems we have implemented the recursive path ordering. Totality checks are performed by Kounalis test. A rough outlook is given about the future work dealing with the application of rewrite techniques in the context of Petri nets.

1. INTRODUCTION

The GRASPIN validation and verification scenario was designed in order to make existing techniques and methods from research area applicable to the working software engineer. It is characterized by two paradigms of modern software development technology:

- Every step in the development of a software product should be equipped with a set of criteria that allows to check the quality of the step; if mathematically rigorous methods are applied the notion verification is used otherwise we call it validation.
- Since currently established validation and verification techniques are connected to massive data (formula) manipulation, a software development system should provide appropriate supporting tools in order to unburden the user and to make the methods practically applicable.

The existing prototype of the GRASPIN workstation supports various validation and verification techniques. Obviously, not every technique or method is applicable at every stage of the software development process. GRASPIN distinguishes different stages requirement definition and analysis (REQ), formal specification (SPEC), and programming (PROG). A second distinction concerns the transition between these

stages. Finally relations between objects manipulated within one stage are considered. Each of these 'problem domains' has specific preconditions in the degree of available mathematical formalism and aims at specific goals in the application of validation and verification methods . The workstation supports these individual requirements (for more details see [1]).

(Semi-) Formal specification is used to reach better understanding of system functionality and behaviour. On the one hand precision is needed for further refinements towards a programming language. On the other hand clear functionality makes communication with customers easier. Powerful and user-friendly editing facilities are a necessary assistance for offering such kind of specifications to the user. The result - well structured, reusable and extensible, precise specifications - is valuable for its own. But, in addition to this advantages formal specification is an important basis for quality assurance approaches. Mainly highly reliable, secure systems require high quality in particular on software programs. For such systems it is of a common understanding that rigorous verification methods are needed. That means documents established during the development process are to be checked or proved as soon as possible. Defined properties representing the required quality must be verified. Using formal specification gives the possibility to do it and to do it early in the life cycle of program development.

In this paper we deal with verification of development steps based on term rewriting systems. In GRASPIN the specification objects are defined in the language SEGRAS® [2] which combines the specification methods of algebraic specifications with those of high level Petri nets. Here we consider mainly the algebraic part of SEGRAS called SPEC.

First we give a rough overview of GRASPIN development methodology [3] in order to facilitate the embedding of the two methods MISOP and MCAI. For each method we provide parts of the computer protocol of example sessions in the annex.

The outlook is focussed on the ongoing work concerning nonsequential systems specified with Petri nets.

The system is available on a SIEMENS AI-machine under INTERLISP-D.

2. THE GRASPIN APPROACH OF SOFTWARE DEVELOPMENT AND VERIFICATION

In order to make large and complex software products more reliable and to reduce the costs of software debugging there should be appropriate techniques and tools assisting the programmer during the software production process. While validation methods are more informal and human assisted, the application of formal specification techniques and verification methods which are based on mathematical theories increases the quality of software. Especially the theory of *algebraic specification* methods and the concept of abstract data types have been well elaborated in the past and they have been successfully applied in development of complex software systems. Early fundamental papers concerned with this field of research are published by Zilles [4], Guttag[5], and the ADJ group [6].

According to this, systems are specified in GRASPIN using the rigorous formalism *SPEC* (the algebraic part of SEGRAS) to express the structure and properties of the intended systems. Objects are considered that introduce data and functions in a highly representation-independent fashion. This allows to concentrate on the problem-specific questions and to disregard environment-specific ones. The SPEC language is supplied with a formal semantics and includes notions and concepts for hierarchical structuring and implementation of objects. Since the specification, programming, and verification steps of the software production are separated, it is difficult to prove automatically that the programs perform the specified tasks. Therefore each development step in GRASPIN has to be confirmed by a validation or even a

verification step *by applying a verification method to an algebraic specification or an algebraic implementation.*

The features of *SPEC* allow software design according to the principles of *stepwise refinement* and *verification while developing*, which means that different levels with varying degree of abstraction are established. The most abstract level describes the overall system structure. This description abstracts from most details and especially from implementation and representation features. The abstract level is gradually refined. On the other hand there will be a more concrete level describing the basic features that will be provided for the overall design task. Within the most concrete level there will usually be a subset of the data types that are provided by the implementation language. At this level it is also possible to include user defined and verified data types. Executable versions of these data types will be available in a basic library. Each level is algebraically specified and consists of a hierarchy of specifications. Starting with some basic specifications, more complex ones are constructed on top of already existing ones.

Besides the horizontal structure, which is constructed completely within one abstraction level, there is the vertical implementation structure, which corresponds to the refinements mentioned above. An abstract level is refined to a level where more concrete details are visible. The final goal of software development is to implement the most abstract level in terms of the most concrete ones and to verify the resulting implementation. This overall implementation and verification task is defined in our setting as the composition of corresponding tasks for every refinement step. We have to be aware that this procedure is not always feasible. But if certain conditions are satisfied, then the feasibility is ensured. So, if we concentrate our attention to the correctness of each single step and if we assume that some conditions (described below) hold, then we will end up with a correct overall implementation.

Let us consider a single implementation step. An abstract specification SP_{up} shall be implemented in terms of the more concrete specification SP_{low}. This *algebraic implementation* can be done completely in the algebraic world with all its advantages. There exists a well known procedure [7] where intermediate levels - the type implementation and the operation implementation - are constructed. The advantage is the very systematic and precise way implementations are constructed and verified. *Verification in this approach means to ensure type-protection, OP-completeness, and RI-correctness* (see chapter 4). The satisfaction of these properties can be reduced to consistency and completeness checks, which are performed by the compound verification methods *MISOP* (Method for the check of Important Specification Object Properties) and *MCAI* (Method for the check of the Correctness of an Algebraic Implementation) which are based on Rewrite Rule techniques.

3. SPEC OBJECTS AND THE VERIFICATION METHOD MISOP

In our approach an abstract data type is syntactically described by an algebraic specification or SPEC object and its semantics is given by a special algebra. It consists of six clauses:

spec	*introduces the name of the specification;*
uses	*lists the names of the specifications used by this specification; all symbols of the types, constructors, and defines clauses of used specifications are visible in this specification;*
types	*lists the names of sets; the types, constructors, and defines clause constitute the signature Σ of a specification; an algebra which consists of a set of values corresponding to each type name and a function corresponding to each function name of Σ is called Σ-algebra;*

constructors	*names and arities of functions; terms built from these symbols generate the value domains of the canonical term algebra which is a special Σ-algebra;*
defines	*names and arities of functions which are equationally defined in the "hornes" clause;*
hornes	*equations built from the names of all visible functions; they describe the behavior of the functions listed in the "defines" clause;*

The following sample specification STK describes the abstract data type stack which stores natural numbers:

spec	STK	
uses	BOOLEAN NATURAL	
types	STACK	
constructors	MT :	-> STACK
	PUSH : STACK NAT	-> STACK
defines	POP : STACK	-> NAT
hornes	POP(MT)	= MT
	POP(PUSH(s,n))	= s

From the mathematical point of view, the signature Σ denotes a class of algebras (Σ-algebras), which consists of:

- a set of values corresponding to each element of the ***types*** clause
- and a function corresponding to each function declaration of the ***constructors*** and ***defines*** clause.

Σ-algebras which satisfy the equations of the "hornes" clause of a specification are called *SPEC*-algebras. Not all Σ-algebras own this property. For example in the term algebra T_{STK}, whose value domains are generated by the terms built from the function symbols of the specification STK, the STACK-terms MT and POP(MT) denote different values, because the two terms are not syntactically identical. But for each specification there is another Σ-algebra, the quotient term algebra QTA_Σ, which is obtained from the term algebra by factorizing the terms according to equations and corresponding definition of its functions.
There are four important properties, which justify to take QTA_Σ as the semantics of an abstract data type, syntactically described by an algebraic specification:

1. QTA_Σ is generated by its functions because it is a quotient of terms built from Σ.
2. QTA_Σ satisfies the equations; so QTA_Σ is a SPEC-algebra.
3. QTA_Σ is the only Σ-algebra (up to isomorphism) with the properties one and two.
4. QTA_Σ is initial, which means that there is a unique homomorphism from QTA_Σ to each SPEC-algebra,

Following to the approach of the ADJ group we finally define that an abstract data type is syntactically determined by an algebraic specification and semantically defined by the class of SPEC-algebras isomorphic to QTA_Σ.

Up to now we did not consider the meaning of *correctness* of an algebraic specification SPEC. If there exists a well known mathematical model ***A*** like the algebra of boolean values or natural numbers corresponding to an intended specification of an abstract data type, we could say that ***SPEC*** is correct w.r.t. ***A***, if QTA_{SPEC} is isomorphic to ***A***. We have to construct a suitable model by our own and then to prove that a specification is correct w.r.t. this model.

As stated above, the quotient term algebra QTA_{SPEC} corresponding to a specification SPEC is generated by the function symbols of the signature. In QTA_{SPEC} each equivalence class of terms corresponds to a value in an arbitrary SPEC-algebra.

Instead of handling with equivalence classes it would be nicer to deal with representatives of these classes. A suitable subset of these canonical terms has to contain exactly one term out of each equivalence class and it must satisfy the following two conditions:

- for each term of T_{SPEC} there must be an equivalent canonical term and

- two canonical terms are equivalent if and only if they are identical.

The Σ-algebra generated in this way is called the *canonical term algebra*. It is a SPEC-algebra and it is isomorphic to QTA_{SPEC}.

According to this, dividing the set of function symbols of a signature in two groups leads to a way how the problem mentioned above can be solved. The former group (constructors) contains the symbols which construct the canonical term algebra. The second group (defines) contains the symbols which correspond to functions which operate on the values established by the constructors. Only the functions of the second group must be equationally defined. Then, the task of proving the correctness of a specification SPEC is to show, that QTA_{SPEC} and C_{SPEC} induced by SPEC are isomorphic. This is true, if the equations of SPEC do not induce an equivalence class of QTA_{SPEC} which contains more than one constructor term.

Often, this proof task can be performed automatically by use of rewrite techniques, because the elements of C_{SPEC} define normalized forms for all terms built from the signature of SPEC. The knowledge, how a term can be rewritten to its normalized form, can be gained from the equations and some further information. If there is a possibility induced by the equations of SPEC to rewrite a term to more than one normalized form, then QTA_{SPEC} and C_{SPEC} are proven to be not isomorphic and hence SPEC is not correct in our sense.

Finally we summarize the important questions which must be examined to show that SPEC fulfills the correctness criterion:

1. Completeness, Confluence, Termination

   Is it always possible to rewrite terms, to which an equation of SPEC is applicable, to a normalized form in a finite number of rewrite steps ?
   If not : Is it possible to generate automatically a finite set of new equations satisfying this requirement ?

2. Consistency

   A positive answer to question one leads to :
   Is the normalized form unique ?
   If not : QTA_{SPEC} and C_{SPEC} are not isomorphic.

3. Totality

   Provided that the second answer is positive :
   Is it possible to rewrite an arbitrary term built from the signature of SPEC to its normalized form ? This is true if the functions specified by the defines clause of SPEC are totally defined.

   If yes : QTA_{SPEC} and C_{SPEC} are isomorphic.

In GRASPIN the compound verification method *MISOP* is applied to SPEC objects to give answers to these questions. Moreover, MISOP tries to complete the set of equations. Generally it can not be decided whether a finite set of equations exists satisfying this requirement. So provers can not always succeed in this task.

MISOP has access to the common GRASPIN data base. It fetches the necessary information from there and the computed results are stored there. Especially in the case that a specification can be proven to be correct, MISOP generates an executable

rule system which will be used by MISOP itself and by other tools (f.e. MCAI , REDUCE see Annex) operating in GRASPIN.

The confluence-check, consistency-check, and completion procedure is based on a modified *Knuth-Bendix-Algorithm* (KBA). This version of the KBA performs a confluence test for ground confluence which is stronger than the classical confluence test. This allows us to prove the ground confluence of term rewriting systems where the classical KBA does not terminate. The latter has recently been applied in theories where only the confluence on ground terms of the equational theory is of interest, as in our application scenario. It tries to generate a term rewriting system which is confluent on arbitrary terms. This often leads to cases where it does not terminate because it generates an infinite rule system, even though this infinite system contains a finite ground confluent system. For more details see [8].

A fundamental subtask performed by MISOP while proving confluence and termination is to rewrite terms to its normalized form. This requires a directed proceeding. For this reason, completion procedures operate on the basis of a hierarchical ordering on operation symbols defined by the user. This ordering determines which of the two possible rewrite prescriptions induced by an equation must be taken to reduce a term towards its normalized form. On this way equations become rules. For proving the termination of term rewriting systems, we have chosen the *recursive path ordering* (RPO). This ordering seems to be the most appropriate ordering for applications in *abstract data types*. There are mainly three reasons for taking this ordering:

1. The RPO extends an ordering on function symbols (precedence) to an ordering on terms. In the context of abstract data types the definition of a function symbol usually bases on previously defined function symbols. In our approach a function symbol f introduced in a signature is greater by definition than all symbols $g_1,...,g_n$, which are visible via the uses clause. By the RPO a ground term t which contains a function symbol f is greater than an arbitrary term t', if t' consists only of symbols which are smaller than f. Also, a ground term t is greater than a ground term t' if the greatest function symbol f of t and t' is contained in both terms but the arguments of f in t are greater than the arguments of f in t'.

2. The weight of the argument position of a function symbol f in t can be defined by its status which can be one of multiset status, left or right. The multiset status determines that every argument position has the same weight, for the left status the left most position has the greatest and the right most position has the least weight, for the right status the meaning is accordingly defined.

3. Two terms can be compared relatively efficiently.

The RPO works for many examples but in cases where cyclic rules occur the user is prompted to direct manually.

For checking the totality of the specified functions the Kounalis test [9] has been implemented. The input for this procedure are a confluent and terminating term rewriting system and a set of constructors with their arity. It checks whether every ground term is equivalent to a constructor term. If one of the functions is not totally defined, *MISOP* presents a list of irreducible terms of the form $f(t_1,...,t_n)$ where f is a defined function symbol and $t_1,...,t_n$ are constructor terms. In order to totalize f, the user could supply the hornes clause of the specification with equations of the form $f(t_1,...,t_n) = t_m$ for every listed term where t_m is a constructor term. This test works for arbitrary specifications, if they can be transformed into confluent and terminating term rewriting systems.

4. STEP OBJECTS AND THE VERIFICATION METHOD MCAI

According to the software development philosophy of GRASPIN, systems are specified algebraically by use of SPEC objects. These objects which are on a high level of abstraction are implemented by SPEC objects which are intuitively on a more concrete level, and so on. This mechanism establishes a chain of implementation steps which ends off with an algebraic specification which is on a very low level of abstraction.

A first systematic treatment of algebraic implementations was given in [7]. A clear separation between syntactical, semantical, and correctness aspects of implementations was achieved. On the syntactical level two steps were introduced: type implementation and operation implementation. In this approach the semantics of an implementation step is systematically divided into the steps *restriction* and *identification*. In order to clarify the notion of algebraic implementation let us introduce the following notation:

- SP_{UP} denotes a specification on an abstract level,
- SP_{LOW} denotes a specification on a concrete, less abstract level.

The first step to implement an abstract specification in terms of a concrete one is called *type implementation*. Syntactically this step is constituted by an algebraic specification $TYPE_{IMPL}$ which is semantically an extension of SP_{LOW}. This condition requires that SP_{LOW} is a part of $TYPE_{IMPL}$ and that the reduct of $TYPE_{IMPL}$ corresponding to SP_{LOW} is isomorphic to SP_{LOW}. In more detail $TYPE_{IMPL}$ consists of SP_{LOW}, the renamed sorts of SP_{UP}, the constructors, which generate the new types needed for the implementation task, and optionally function symbols of the new types, which have to be specified by equations.

The goal of this step is to set up the correct data representations to be capable of implementing SP_{UP} later on. A second aim is to rename types to avoid name conflicts. For the construction of the appropriate data representations the usual type constructors, known from programming languages like records, arrays, etc., might be chosen.

The purpose of the second step, called *operations implementation*,is to realize the functions denoted by SP_{UP} by functions denoted by SP_{LOW}. Formally the renamed function symbols of SP_{UP} are added to $TYPE_{IMPL}$ together with equations, describing the behavior of the corresponding functions in terms of $TYPE_{IMPL}$.

Until now, only the syntactical aspects of an implementation step have been presented, so now we have to look at the semantics of them. After having extended the lower level data type by data and functions to perform the activities of the upper level data type, we are now going to forget all data and functions which are not corresponding to types and functions induced by SP_{UP} in a first *restriction step*, called FORGET. Because of the possibility, that there still remain data which will never appear during applications of upper level functions, we have to forget all these data, not reachable by upper level function applications, in a second *restriction step*, called REACH. More formally, these two steps can be characterized as follows:

- result(FORGET) can be looked as a Σ-algebra corresponding to the signature of SP_{UP}

As a consequence, there exists a unique homomorphism $r : T_{SP_{UP}} \rightarrow result(FORGET)$

- The reachable data can be characterized by $result(REACH) = r(T_{SP_{UP}})$

The *restriction* steps did not take the equations of SP_{UP} into account. This is finally done in the *identification* step by considering the congruence relation, which is induced by the equations of OP_{IMPL}, in which all function symbols are renamed by their corresponding function symbols of SP_{UP}. Let ID_{IMPL} be a quotient,which is

constituted by REACH factorized according to this congruence relation. Thereby the homomorphism r is transformed into a homomorphism between the corresponding quotient term algebras : r' : $T_{SP_{UP}}$ -> ID_{IMPL}.

After having analyzed the semantics of an algebraic implementation in more detail, the question arises: *When is an algebraic implementation correct?*
Fundamentally we require that all constructive steps do not disturb the original abstract data types. In the case of *type implementation* this requirement is called *type protection*. A similar requirement must be imposed onto the step *operation implementation*. This must be an enrichment of type implementation. Normally the equations, that are introduced by the operation implementation step, will define the new functions explicitly and will thus not raise inconsistencies. A problem in this context is, that computations performed by the new functions are not reducible to computations performed by functions and type implementation. This requirement is called OP-completeness. Whenever the new operations are totally defined, this requirement is automatically satisfied. But in general, the operations might be implicitly defined and hence the same checks as in type protection become necessary.

But what do we require beyond this? Simply asking for satisfaction of equations of SP_{UP} in OP_{IMPL} would be too strong. A closer look at this criterion shows, that most practical implementations would not satisfy this requirement. The reason why is, that in most practical applications an abstract data is represented by more than one data in the concrete level. A solution of this problem is to identify firstly the data on the lower level according to the equations in the upper level. This has been done in the identification step above. Secondly, we have to require that distinct data in the upper level must be represented by distinct data in the lower level. This notion of correctness of algebraic implementations is called RI-correctness. In this context R stands for the *restriction step* and I for the *identification step*.

In GRASPIN the compound verification method *MCAI* is applied to step objects, which define algebraic implementation steps, in order to prove their correctness. *MCAI* is based on the same verification techniques as MISOP and in addition it uses the tools *SEEC* (Syntactical Extension Enrichment Checker) and *HPC* (Homomorphism Property Checker). *SEEC* tries to show by use of syntactical criterions, that a specification is an enrichment respectively an extension of another specification. HPC tries to show, that a specification HOM, which maps turns of $T_{OP_{IMPL}}$ to terms of $T_{SP_{UP}}$ together with an algebraic implementation step induce a homomorphism between $QTA_{OP_{IMPL}}$ and $T_{SP_{UP}}$. Again HOM is based on rewrite techniques. The algorithm *MCAI* works as follows:

a) *type protection check*: Is $TYPE_{IMPL}$ an extension of SP_{LOW} ?

a1) application of SEEC

- SEEC succeeds: branch to b
- SEEC fails:

a11) Induces $TYPE_{IMPL}$ a confluent, terminating, consistent rule system ?
- Success: branch to b
- Failure: branch to failure-exit

b) *OP-completeness check*: Is OP_{IMPL} an enrichment of $TYPE_{IMPL}$?

b1) application of *SEEC*
- *SEEC* succeeds: branch to c
- *SEEC* fails:

b11) Induces OP_{IMPL} a confluent, terminating, consistent rule system ?
- Success: branch to c
- Failure: branch to failure-exit

c) *RI.correctness check*: Is ID_{IMPL} consistently specified w.r.t. OP_{IMPL} ?

c1) application of SEEC
- *SEEC* succeeds: branch to success-exit
- *SEEC* fails:

c11) Induces ID_{IMPL} a confluent, terminating, consistent rule system ?
- Success: branch to success-exit
- Failure: application of *HPC* to HOM
  - Success: branch to success-exit
  - Failure: branch to failure-exit

failure-exit) The algebraic implementation could not be proven to be correct.

success-exit) The algebraic implementation could be proven to be correct.

An example for the MCAI method is presented in the annex.

5. OUTLOOK

As an interesting question, and an area of current and future research in GRASPIN, we ask for a suitable concept to make algebraic methods, such as algebraic specifications, ADT's, term-rewriting and the like, applicable to describe and analyze asynchronous and concurrent processes. Such processes are usually described by Petri nets, a formalism, whose expressive power has steadily increased throughout the years. Different approaches, treating different levels of Petri nets, are pursued to describe the corresponding nets by objects of the semigraphical specification language SEGRAS respectively by the SPEC-subset of SEGRAS mentioned above, to derive a term rewriting system that can be used as an effective means to check properties of the underlying system specified, such as lifeness, deadlockfreeness, fairness, reachability and/or proper termination.

As a first level, Predicate/Event nets are considered. To work with such nets means for example to construct firing rules, to simulate or analyze such nets. Analyzing nets implies proving the existence or nonexistence of forward or backward conflicts, detecting deadlocks or to test the reachability of markings. In the GRASPIN environment nets appear as graphs yielding a scheme defined with respect to a partial abstract data type, the data that are produced, consumed or manipulated are terms of the data type in question. S- and T-elements, which represent the components of the nets, are introduced as partial functions, the program steps have to be synthesized on one hand of firing rules, defined by the topological environment of the transitions, and on the other hand from the variable substitutions which are attached as labels to the arcs. The fact that tokens can actually lie on S-elements is expressed by subfunctions of these S-elements which are just defined for the actual tokens or data respectively. These subfunctions can be extended or restricted, they can thus be considered as functional terms created and manipulated by generation and cancellation operations, using the empty and the generation operations as constructors. The marking of nets are then functions mapping the S-element on the actual functional terms. In a nearby and constructive way the firing rules for the T-elements can then be described using these operations.

The rewrite system, which is provided for the GRASPIN environment, deals only with totally defined operations and unconditional equations. Thus partial operations as discussed above have to be excluded or to be extended to total operations by introducing error elements for the sorts in question. Moreover conditional equations have to be replaced by introducing appropriate if-then-else operators as well as a specification BOOL yielding an initial two valued model for the booleans. The markings, which represents actual states, can be considered as terms of the specification. By unification methods of the rewrite system the firing rules for the markings are transformed into rewrite rules between these terms; by this method the

behaviour of the net can be described completely by abstract data types together with a set of rules, hence allowing the application of the usual rewrite methods and criteria of rewriting.

The second level considered, the Place/Transition nets, is of less expressive power. Here, the data to be manipulated are essentially restricted to the set of natural numbers. These nets can be translated in a straight forward manner into an algebraic specification yielding a term-rewriting set, whose usefulness, also with regards to efficiency and feasibility, has yet to be examined.

There are reasons to be hopeful that an algebraic treatment, in the sense indicated above, is possible at least for low-level Petri nets; however, questions arising when dealing with high-level Petri nets are yet to be answered.

NOTES AND REFERENCES

® SEGRAS is a registered trademark of GMD

⊙ The reason for the distinction of constructor and defined functions is the automatization of correctness proofs. The prover derives contradictions in equational specifications from syntactical comparisons of constructor terms.

[1] Dehm,B. et al : Description of V&V Methods, GRASPIN Technical Paper SIE20/2, Sankt Augustin: GMD/Olivetti/Siemens, 1985

[2] Krämer,B.: SEGRAS The GRASPIN Specification Language Preliminary Reference Manual,GRASPIN Technical Paper SIE20/2,Sankt Augustin: GMD/Olivetti/Siemens, 1986

[3] Dehm,B.,Haensse,Th.: The GRASPIN Approach to Software Validation and Verification, in ESPRIT '86: Results and Achievements (North-Holland), 1987

[4] Zilles, S.N.: Algebraic specification of data types. Project MAC Progress Report 11, MIT 1974, 28-52

[5] Guttag, J.V.: The specification and application to progamming languages of abstract data types. Ph.D. Thesis, University of Toronto, 1975

[6] Gougen, J.A., Thatcher, J.W., Wagner, E.G.: An initial approach to the specification, correctness, and implementationof abstract data types,IBM Research Report RC 6487, 1976

[7] Ehrig, H., Kreowski, H.-J., Padawitz, P.: Algebraische Implementierung Abstrakter DatentypenBericht-Nr. 79-3, Technische Universität Berlin

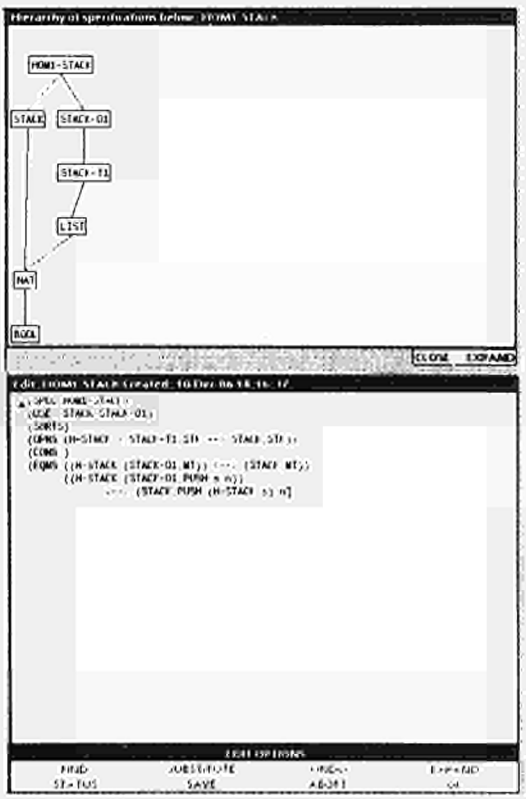
[8] Göbel, R.: Ground Confluence SEKI-REPORT SR-86-18 University of Kaiserslautern, 1987

[9] Kounalis, E.: Completeness in data type specifications In theProc. EUROCAL 85 Springer Lecture Notes Linz 1985

[10]Sommer, W.:Description of the current state of theVerification Module (VM) of the GRASPIN workstation ,University of Kaiserslautern, 1987

ANNEX

The SHOW command of the RRLab gives an overview over the specification hierarchy of the example HOM1-STACK (see first window below).
The STACK-Module is implemented via LIST-Module. HOM1-STACK links the specification STACK with the abstract implementation (see EDIT window below).



For HOM1-STACK we are going to apply MISOP. A SERVICE command customizes the information put on the session protocol.

MISOP
Method for the checking of Important SPEC Object Properties.
Application of the RRLab.

Checking specification HOM1-STACK
Computing the transitive precedence closure of operations
used by specification HOM1-STACK
..... done.
New rule generated:
HOM1-STACK[1]: H-STACK(MT) --> MT

New critical overlappings:
The rules :
HOM1-STACK[1]: H-STACK(MT) --> MT
and :
STACK-O1[1]: MT --> I(NIL)
generate at position (1) a critical pair.
The term :
H-STACK(MT)
can be reduced via the first rule to :
MT
and via the second rule to :
H-STACK(I(NIL))

New rule generated:
HOM1-STACK[1], STACK-O1[1] = > HOM1-STACK[2]: H-STACK(I(NIL)) --> MT
New rule generated:
HOM1-STACK[3]: H-STACK(I(C(a,P(b)))) --> PUSH(H-STACK(b),a)

New critical overlappings:
The rules :
HOM1-STACK[3]: H-STACK(I(C(a1,P(b1)))) --> PUSH(H-STACK(b1),a1)
and :
STACK-T1[1]: P(I(a1)) --> a1
generate at position (1 1 2) a critical pair.
The term : H-STACK(I(C(a1,P(I(b1)))))
can be reduced via the first rule to :
PUSH(H-STACK(I(b1)),a1)
and via the second rule to :
H-STACK(I(C(a1,b1)))

New rule generated:
HOM1-STACK[3], STACK-T1[1] = >
HOM1-STACK[4]: H-STACK(I(C(a,b))) --> PUSH(H-STACK(I(b)),a)

Removed rules:
HOM1-STACK[3]: PUSH(H-STACK(I(P(b))),a) --> PUSH(H-STACK(b),a)

New rule generated:
HOM1-STACK[5]: PUSH(H-STACK(I(P(a))),b) --> PUSH(H-STACK(a),b)

New critical overlappings:
The rules :
HOM1-STACK[5]: PUSH(H-STACK(I(P(a1))),b1) --> PUSH(H-STACK(a1),b1)
and :
STACK-T1[1]: P(I(a1)) --> a1
generate at position (1 1 1) a critical pair.
The term : PUSH(H-STACK(I(P(I(a1)))),a1)
can be reduced via the first rule to :
PUSH(H-STACK(I(a1)),a1)
and via the second rule to :
PUSH(H-STACK(I(a1)),a1)

The system is confluent and terminating!

Specification HOM1-STACK is consistent.
The resulting rule system has been saved.

Checking the totality of operations :

Checking the left hand side of rule :
HOM1-STACK[5]: PUSH(H-STACK(I(P(a))),b) --> PUSH(H-STACK(a),b)
The rule defines a constructor operation.

Checking the left hand side of rule : HOM1-STACK[1]: H-STACK(MT) --> MT
The rule doesn't contribute to the definition of a total rule system.

Checking the left hand side of rule :
HOM1-STACK[1], STACK-O1[1] = > HOM1-STACK[2]: H-STACK(I(NIL)) --> MT
The rule contributes to the definiton of a total rule system.

Checking the left hand side of rule :
HOM1-STACK[3], STACK-T1[1] = >
HOM1-STACK[4]: H-STACK(I(C(a,b))) --> PUSH(H-STACK(I(b)),a)
The rule contributes to the definiton of a total rule system.

The following rules do not contribute to a total operation definition :
HOM1-STACK[1]: H-STACK(MT) --> MT
HOM1-STACK[5]: PUSH(H-STACK(I(P(a))),b) --> PUSH(H-STACK(a),b)
In the following they are treated as inductive lemmata.
Checking totality of HOM1-STACK.H-STACK .... done.
Operation HOM1-STACK.H-STACK is totally defined.

All operations are totally defined.

The "STATUS" command of EDIT window gives us an overview about the stored MISOP results:

```
Rewrite-Rule-Laboratory -- VERSION 3.0

Status report for specification HOM1-STACK :

CREATED     : 31-Jul-87 10:53:05

PARSED      : 31-Jul-87 10:53:39

COMPLETED   : YES

CONFLUENT   : YES

The following rule system has been generated :
HOM1-STACK[5]: PUSH(H-STACK(I(P(a))),b)
 -->  PUSH(H-STACK(a),b)
HOM1-STACK[1]: H-STACK(MT)
 -->  MT
HOM1-STACK[1], STACK-O1[1] = >
HOM1-STACK[2]: H-STACK(I(NIL))
 -->  MT
HOM1-STACK[3], STACK-T1[1] = >
HOM1-STACK[4]: H-STACK(I(C(a,b)))
 -->  PUSH(H-STACK(I(b)),a)

TERMINATING : YES

TOTAL       : YES

CONSISTENT  : YES
```

As example for MCAI we are going to check the implementation step STACK1.
The step object "STACK1" (see below) provides the abstract implementation chain and the defined morphisms for sorts and operations.

```
Edit Step : STACK1 Created: 3-Dec-86 14:39:32
((STEP STACK1)

 (USE )

 (A-IMPL (STACK STACK-01 STACK-T1 LIST))

 (MORPHISM (SORTS-MAPPING (STACK,STK STACK-T1,STK))
           (OPS-MAPPING (STACK,MT STACK-01,MT)
                        (STACK,POP STACK-01,POP)
                        (STACK,PUSH STACK-01,PUSH)
                        (STACK,TOP STACK-01,TOP)))

 (C-IMPL XXX)
)
```

MCAI
Method for the checking of the Correctness of an Algebraic Implementation step.

Checking step STACK1

Type protection check :
Is specification STACK-T1 a syntactical extension or enrichment of LIST ?
Application of SEEC ... No success.

Do the equations of STACK-T1 induce a confluent, terminating, total and consistent Rewrite Rule System ?
Application of CTTCC ...

CTTCC result : The equations of STACK-T1 induce a confluent, terminating, total and
consistent Rewrite Rule System.

Op-completeness check :
Is STACK-O1 completely specified w.r.t. STACK-T1 ?
Specification STACK-O1 is a syntactical enrichment of specification STACK-T1.
Application of SEEC ... Success.

RI-correctness check :
Is IDIMPL-STACK1 (that is STACK-O1 together with the adapted equations of STACK)
consistently specified w.r.t. STACK ?
Application of SEEC ... No success.

NOTE: The SEEC cannot be be successfull at this point, because the syntactical extensionlenrichment property is (at the moment) defined only between those specifications, that are in a USE-relation.

Do the equations of IDIMPL-STACK1 induce a confluent, terminating and consistent Rewrite Rule System ?
Constructing IDIMPL-STACK1 ... done.
Application of CTTCC ...
Computing the transitive precedence closure of operations
used by specification IDIMPL-STACK1
..... done.

New rule generated:
IDIMPL-STACK1[1]: POP(MT) --> MT

New critical overlappings:
The rules :
IDIMPL-STACK1[1]: POP(MT) --> MT
and :
STACK-O1[1]: MT --> I(NIL)
generate at position (1) a critical pair.
The term : POP(MT)
can be reduced via the first rule to : MT
and via the second rule to : POP(I(NIL))

New rule generated:
IDIMPL-STACK1[2]: I(P(a)) --> a

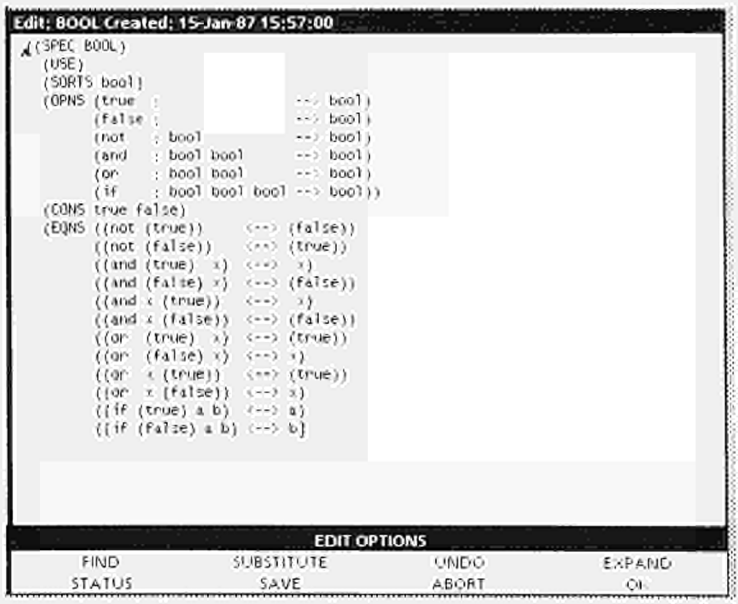
New critical overlappings:
The rules :
IDIMPL-STACK1[2]: I(P(a1)) --> a1
and :
STACK-T1[1]: P(I(a1)) --> a1
generate at position (1) a critical pair.
The term : I(P(I(a1)))
can be reduced via the first rule to : I(a1)
and via the second rule to : I(a1)

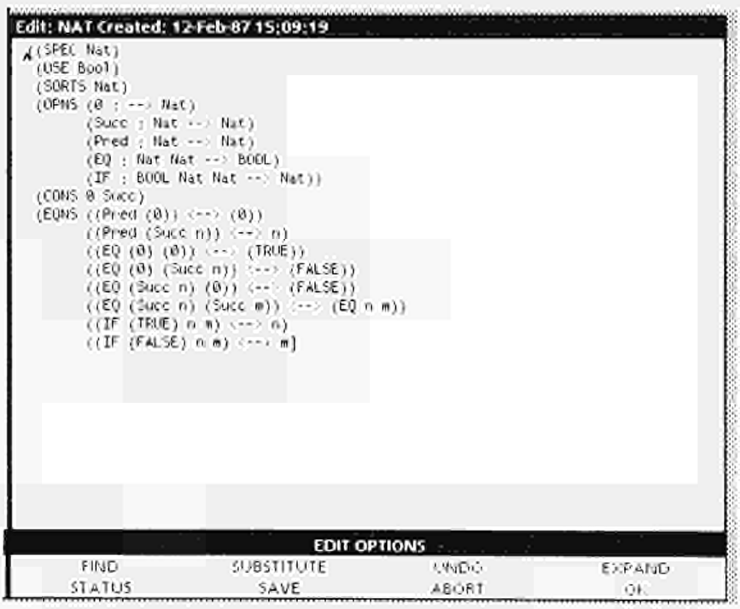
The system is confluent and terminating!
Removing IDIMPL-STACK1 ...

CTTCC result : The equations of IDIMPL-STACK1 induce a confluent, terminating and consistent Rewrite Rule System.

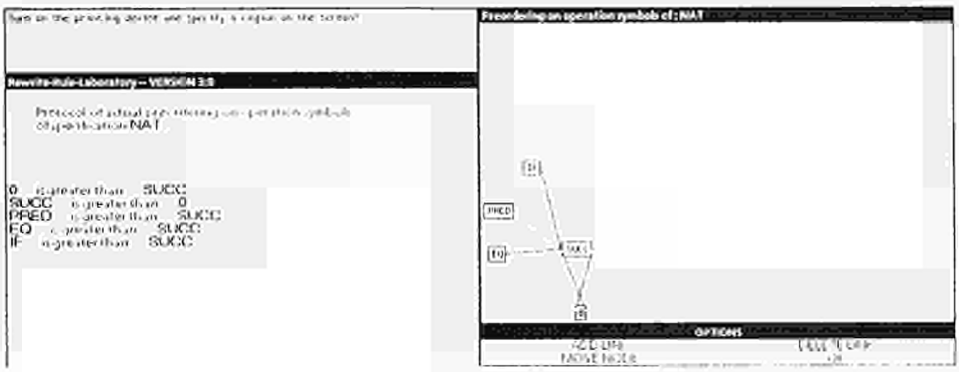
Our last example will be the execution of terms invoked by the REDUCE command.
The following term
(if (and (eq (pred(succ(succ n)))(succ n))(TRUE))(succ n)(0))
should be reduced .It means that
(pred (succ (succ n)) = (succ n) $\wedge$ TRUE) $\equiv$ TRUE $\Rightarrow$(succ n)
or that
(pred (succ (succ n)) = (succ n) $\wedge$ TRUE) $\equiv$ FALSE $\Rightarrow$(0)
holds.

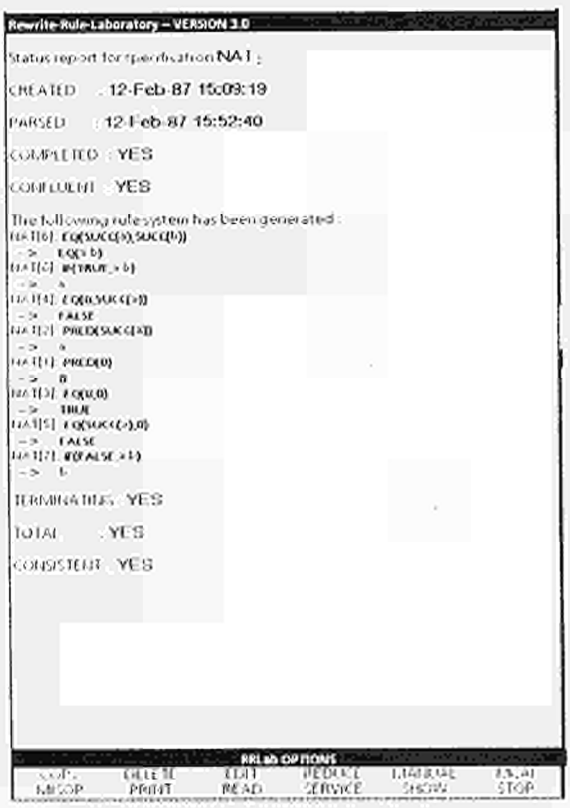
In our example we only use rules of the specification BOOL and NAT (see below).





The userdefined preordering on operation symbols of NAT, and the reports on STATUS and REDUCTION are given in the following windows.






Rewrite-Rule-Laboratory -- VERSION 3.0
Execution report of the REDUCE command :
Select the name of the specification at the top of the rule system :
Note : Only the rules of the hierarchy of specifications below
NAT are used in reduction steps.
Each operation symbol must be prefixed by the name
of the corresponding specification.
To stop the reduction process type in 'b' or 'B'
No type checking will be performed before the reduction.
Enter the term or equation to be reduced :
(nat, if(Bool,and(nat,eq(nat,pred(nat,succ(nat,succ n))))(nat,succ n))
(Bool true))(nat,succ n)(nat,0))
Reduction by rule :
Bool[4]: AND(x,TRUE)
-> x
to term :
IF(EQ(PRED(SUCC(SUCC(N))),SUCC(N)),SUCC(N),0)
Reduction by rule :
Nat[-]: PRED(SUCC(a))
-> a
to term :
IF(EQ(SUCC(N),SUCC(N)),SUCC(N),0)
Reduction by rule :
Nat[6]: EQ(SUCC(x),SUCC(b))
-> EQ(x,b)
to term :
IF(EQ(N,N),SUCC(N),0)
Reduction stopped. Term :
IF(AND(EQ(PRED(SUCC(SUCC(N))),SUCC(N)),TRUE),SUCC(N),0)
reduced to term :
IF(EQ(N,N),SUCC(N),0)
RRLab OPTIONS
COPY
DELETE
EDIT
REDUCE
MANUAL
MEAT
MKSP
PRINT
READ
SERVICE
SHOW
STOP

Project No. 1072c

TOWARDS RELIABLE COMPUTING

J. Kok and D. T. Winter

Centrum voor Wiskunde en Informatica,
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

M. J. Erl and G. S. Hodgson

Numerical Algorithms Group Ltd, NAG Central Office, Mayfield House,
256 Banbury Road, Oxford, OX2 7DE, England

1. INTRODUCTION

Floating-point operations are characterised by the rounding (or truncating) of the result to the finite precision of the mantissa of the floating-point representation.

Whilst these floating-point operations may be accurate to 1/2 or 1 in the least significant bit of the mantissa, it is the accumulation of such rounding errors after a large number of operations which can be unpredictable. Even more serious is the danger of cancellation with floating-point addition or subtraction (10.0**15 + 1.0 – 10.0**15 may well yield zero in floating-point arithmetic).

These dangers can be avoided by the use of a fixed-point accumulator to hold intermediate results in a series of calculations and then rounding once only at the end of the calculation. An accurate scalar product, which is an integral part of many vector and matrix operations, can thus be calculated with full accuracy.

This form of accurate arithmetic was first proposed by *Kulisch* and *Miranker* [6] and its realisation in Ada[1] by *Klatte, Ullrich* and *Wolff von Gudenberg* [5]. As part of the ESPRIT project DIAMOND [4] this embedding in Ada has been refined with particular regard to its efficient realisation and transportability across a range of Ada compilation systems.

The operations provided are the usual arithmetic primitives

$$+, -, *, /,$$

together with an accurate scalar product. These operations are provided in real arithmetic, complex arithmetic and for real and complex intervals.

For the experienced programmer access is also provided to the fixed-point accumulator, together with directed roundings (above, below or to the nearest value).

---

[1] Ada is a registered trademark of the US Department of Defense AJPO

With these accurate operations embedded in Ada, it is possible to develop reliable numerical libraries in Ada whose accuracy is guaranteed. The DIAMOND project is investigating the development of such packages in the areas of:

a) solution of linear equations and inversion,
b) calculation of eigenvalues and eigenvectors,
c) zeros of polynomials,
d) analysis of sparse systems,
e) analysis of systems of non-linear equations,
f) numerical integration.

We consider the embedding in two parts:
a) the user interface,
b) the primitive underlying operations.

In the next Section we will discuss the accurate floating-point facilities offered to the user, as they have been designed and implemented in Ada, while in the subsequent Sections we will concentrate on the primitive operations.

2. THE ADA USER INTERFACE FOR SCIENTIFIC COMPUTATION

Section 2.1 gives a survey of the accurate arithmetic to be implemented, and sections 2.2 and 2.3 explain how the activity takes benefit of the use of the language Ada and describe the achieved result.

2.1. ACCURATE ARITHMETIC

In early programming languages a precise definition of the floating-point operations was absent. Floating-point arithmetic was provided by the manufacturers in many different, and even peculiar ways, along with the representation of floating-point numbers. Not well-specified were areas like accuracy of operations, representation boundaries, and overflow and underflow. It was probably assumed that users were familiar with the properties of the floating-point arithmetic they employed, and that they would take appropriate measures for the unstable cases where the inaccuracies of the arithmetic would spoil the results of their computations. Experience over the past decades has taught that only few users are willing to pay such attention to obtaining known error bounds for the results of computations and to programming computations in such a way that the error bounds are acceptable.

With the adoption of the IEEE Standard 754 for binary floating-point arithmetic [1] it should not be a problem anymore to obtain hardware with the desirable floating-point arithmetic characteristics. However, this standard is concerned with the required accuracy of only the basic operations (+, −, *, /), stating that for these operations round-off errors should not exceed the value of the least significant bit of the result (the standard further addresses the subjects of minimal precision of the floating-point representations, boundaries with overflow and underflow behaviour, and choices for the rounding modes). But this leaves unsolved the problem of the growth of errors for complicated floating-point computations and the derivation of known error bounds for their results. Clear and

easy-to-understand examples with an unacceptable growth of the rounding errors can be found, e.g., in [8].

In the mathematical theory developed by Kulisch & Miranker (see, e.g., [6], [7], [8]) it is described how the available floating-point arithmetic can be supplemented to provide reliable tools for the scientific computing community. Mathematical computations are defined (apart from *integer arithmetic*) with numbers taken from either the mathematical set of *real* numbers (**R**) or that of *complex* numbers (**C**), and with *intervals*: *real intervals* that consist of two boundaries (INF and SUP) which are real numbers, and *complex intervals* whose boundaries are two complex numbers (the sets are called *I***R** and *I***C**). Further, for these 4 number spaces the set of all vectors and the set of all matrices are defined, yielding the additional spaces (with *V* & *M* for vector and matrix, respectively) *V***R**, *M***R**, *IV***R**, *IM***R**, *V***C**, *M***C**, *IV***C**, *IM***C**.

The shortcoming of computer arithmetic now lies in the fact that floating-point operations take place in subsets of these 12 spaces only, i.e. in finite subsets consisting of machine-representable numbers only (*number* in the general sense). For the names of *these* 12 subsets we replace **R** and **C** by *R* and *C*, and the relations are now shown in table 1 below.

mathematical space		*set of machine representations*
R	$\supset$	*R*
*V***R**	$\supset$	*VR*
*M***R**	$\supset$	*MR*
*I***R**	$\supset$	*IR*
*IV***R**	$\supset$	*IVR*
*IM***R**	$\supset$	*IMR*
C	$\supset$	*C*
*V***C**	$\supset$	*VC*
*M***C**	$\supset$	*MC*
*I***C**	$\supset$	*IC*
*IV***C**	$\supset$	*IVC*
*IM***C**	$\supset$	*IMC*

Table 1.

Computations taking place in the spaces of machine-representations are obviously required to approximate the corresponding ideal computations as well as possible. This is solved by requiring accurate arithmetic for the normal, i.e. scalar operations ($+, -, *, /$), and also for several operations involving vectors and matrices, in particular the *inner product* $\sum_{i=1}^{n} v_i * w_i$ of two vectors v and w (also occurring in vector-matrix and matrix-matrix multiplications).

For these operations accurate rounding of the exact result to the nearest representable number (by the mapping $\square$), and rounding downward (by ∇) and upward (by Δ) must be provided for the **R** and **C** spaces, with rounding outward (by $\Diamond$) for computations involving intervals.

This set of accurate operations was first available in the Pascal extension Pascal-SC [3] developed in Karlsruhe. The present paper describes the embedding of accurate arithmetic in the modern programming language Ada [2].

2.2. USING ADA FOR IMPLEMENTING ACCURATE ARITHMETIC

For the embedding of accurate arithmetic Ada was considered very suitable because of its design criteria of abstraction, portability and readability, and because of the international standardisation activities which have accompanied its introduction. The Ada language features for producing components of large software libraries can conveniently be used for designing and implementing operators for basic arithmetic satisfying requirements of general usefulness, flexibility, ease of use, and transportability, while allowing efficient execution as much as possible.

With Ada there is no urgency to deviate from the language definition, since the necessary features are readily available. Further, the validation procedure for Ada compilers and environments guarantees a close (though not perfect) conformance of compilers to the language definition, which enables truly portable programming. This makes Ada a suitable vehicle for implementing the embedding of accurate arithmetic.

2.3. THE USER INTERFACE

For the top-level package of accurate arithmetic facilities, the DIAMOND project objective is to satisfy the natural requirements of programmers of mathematical computations.

Such users want to obtain operators which enable them to program expressions that resemble very much the original mathematical formulae, even for mathematical spaces of complex numbers or intervals, and for linear spaces thereof. Obviously, this practice is allowed in Ada through operator declarations and the overloading principle, although the introduction of new symbols for operators is not allowed. The primary requirement to accommodate mathematically-oriented users is therefore readily satisfied.

However, these users may also want to control the rounding mode (i.e. choosing the rounding direction) for the basic arithmetic operations on real values. Since some distinction between different operations is necessary and Ada does not allow the introduction of new symbols for new operators, *functions* (instead of *operator* declarations) are provided in particular for the floating-point and complex types.

Another requirement, which can be met easily, is the possibility to cope with a user-defined choice of the floating-point type underlying all number spaces, viz. through the declaration

```
type REAL is digits N [ range A .. B ];
```

The Ada solution for this requirement serves the portability goal as well.

It is a natural requirement that data structure details will be hidden. This is applied to the data structures needed for temporarily storing results of higher accuracy (for the user these are **private** types). For the *mathematical* data types this is not required, but it was decided that the types for *intervals* and *complex intervals* should also be **private** to prohibit the construction of illegal representations for values of these types. The data structures for temporarily storing results of higher accuracy are actually made *limited* **private**, to prohibit the use of the "=" operator which might give a misleading result.

The project result is a set of Ada packages with a clear separation between the facilities offered to the user and the (exchangeable) bodies containing technical bits.

The user facilities are presented in a *generic package*, GENERIC_SCIENTIFIC_COMPUTATION, which has the following Ada skeleton:

```
generic

   type FLOAT_TYPE is digits <>;
   --
   type COMPLEX_TYPE is private;

   -- Several additional generic parameters

package GENERIC_SCIENTIFIC_COMPUTATION is

   package REAL_ARITHMETIC is
   --...
   end REAL_ARITHMETIC;

   package COMPLEX_ARITHMETIC is
   --...
   end COMPLEX_ARITHMETIC;

   package INTERVAL_ARITHMETIC is
   --...
   end INTERVAL_ARITHMETIC;

   package COMPLEX_INTERVAL_ARITHMETIC is
   --...
   end COMPLEX_INTERVAL_ARITHMETIC;

end GENERIC_SCIENTIFIC_COMPUTATION;
```

The subpackages relate to the 4 number spaces (see Table 1), and they contain the accurate operations with the different rounding modes together with extra facilities for temporarily computing in higher precision.

Particular attention has been paid to:

- *general usefulness*, like providing all necessary operations, but also the flexibility for using the facilities, and portability,

- *Readability*: structure of the package, completeness of the definitions, and hiding of irrelevant details,

- Requirements arising from the package *implementation* task, where aspects of efficiency and feasability influence the specification.

3. REQUIREMENTS OF THE PRIMITIVE ACCURATE OPERATIONS

The package GENERIC_SCIENTIFIC_COMPUTATION (G_S_C) gives a modular design for the user interface to the embedding of accurate arithmetic with four subpackages for real, complex, real interval and complex interval arithmetic. It uses as an auxiliary the package GENERIC_ACCURATE_ARITHMETIC (G_A_A) which defines the *primitive accurate arithmetic types and operations*, that is the declarations of higher accuracy types and operations on such types; most aspects of the implementation of such types can be isolated in the private part of G_A_A.

For example the type DOT_PRECISION used to represent a fixed-point accumulator is private; the user is therefore not aware of the details of its representation, but must access the accumulator using the visible operations which we provide.

The types and operations defined in G_A_A are used to implement the subpackage GENERIC_SCIENTIFIC_COMPUTATION.REAL_ARITHMETIC; it is then possible to implement the other subpackages for complex and interval arithmetic in terms of reals, or more directly (and efficiently) using the operations of G_A_A itself.

The task of implementing the primitive accurate arithmetic operations in G_A_A is neither simple nor straightforward. Our desire is to provide a portable implementation whilst not sacrificing efficiency. These are sometimes conflicting aims.

Because G_A_A is written as a separate package, we cannot prohibit users accessing G_A_A directly. Indeed the sophisticated user may wish to use G_A_A on its own in order to construct alternative forms of G_S_C or to enhance its functionality in some areas. Provided G_A_A is sufficiently big (functionality-wise) and small enough (storage-wise) it can be used as a stand-alone entity.

Both of the packages G_S_C and G_A_A are generic so that different precisions of arithmetic can be provided. One obvious generic parameter is the user's type FLOAT_TYPE used to instantiate G_S_C and passed further to G_A_A; or directly to G_A_A. Since there will be no need for simultaneous use of G_S_C and G_A_A there is no danger of instantiating with different floating-point types.

Environmental parameters are provided in the Ada language through attributes. For each predefined floating-point type FLOAT_TYPE the attributes

MACHINE_MANTISSA, MACHINE_RADIX, MACHINE_EMAX, MACHINE_EMIN

are sufficient to establish the framework for the accurate arithmetic:

- availability of higher precision floating-point types to perform double_precision arithmetic

- possible fixed-point types suitable for DOT_PRECISION (only one is guaranteed)

- characteristics of integer types (when (almost always) fixed point types are not suitable for the implementation)

and based on those:

- the length of the long accumulator

- conversion rules between a user's floating-point types and their internal representation (used by the accurate operators).

Also available is the attribute MACHINE_ROUNDS to determine the suitability of real "+","-","*" and "/" for their accurate counterparts ADD, SUBTRACT, MULTIPLY and DIVIDE.

We thus have a portable implementation of the primitive operations which adjusts automatically to the different hardware environments.

4. SCALAR PRODUCT

4.1. LONG ACCUMULATOR

The long accumulator is the fundamental extended precision type (DOT_PRECISION) used in G_A_A. We are free to choose the internal representation for the DOT_PRECISION type because it is declared as private; details of its representation are not therefore available to users. Our choice is between three forms of representation:

```
type DOT_PRECISION is -- using fixed-point type
   delta 10**(-2*(REAL'EMAX+REAL'DIGITS))
   range -10**( 2*(REAL'EMAX+REAL'DIGITS))
      .. 10**( 2*(REAL'EMAX+REAL'DIGITS));-- Klatte [5]

type DOT_PRECISION is -- using floating-point type
   record LS: LONG_STORAGE;
          START, FINISH : INTEGER ;
   end record;
   -- where --
   -- dmin = 2 * ('machine_emin/'machine_mantissa-1)
   -- dmax = 2 * 'machine_emax/'machine_mantissa+1
type LONG_STORAGE is
   array(DMIN..DMAX+GUARD_ELEMENTS) of real;

type DOT_PRECISION is -- using integer type
   record
```

```
    START : INTEGER := ENVIRONMENT.ACCU_END ;
        FINISH: INTEGER := ENVIRONMENT.ACCU_FIRST;
        VALUE : INTEGER_VECTOR:=(others=>0);
    end record;
    -- where for example --
    INTEGER_VECTOR is array
        (ENVIRONMENT.ACCU_FIRST..ENVIRONMENT.ACCU_END) of INTEGER;
```

In all cases the size of the type is determined by attributes, although in the last case (the integer representation) the calculation needs to be done in an auxiliary package ENVIRONMENT to satisfy language restrictions.

We believe the last form is likely to be most suitable for the largest number of machines, although the other representations can be substituted in special cases without the user being aware of the chosen representation.

The scalar product is the fundamental operation of the accurate arithmetic. It can be simply expressed in terms of two subprograms: DOT_ADD to accumulate the product in extended precision and ROUNDS to convert back to floating-point form.

```
function "*"(V,W : in VECTOR_TYPE)return FLOAT_TYPE is
    C          : DOT_PRECISION;
    DOWN,UP,NEAR : FLOAT_TYPE;
begin
    DOT_ADD(C, V, W);
    ROUNDS (C, DOWN,UP,NEAR);
    return NEAR;
end "*";
```

where VECTOR_TYPE is an array of FLOAT_TYPE (both are generic parameters of G_A_A).

This scalar product specification does not reveal to the user the existence of an intermediary DOT_PRECISION accumulator. However, it is expected that the user will have (limited) access to this maximal-accuracy type (mainly via the procedure DOT_ADD); and so he must be able to determine all 3 ROUNDingS after any attempt to change the contents of C.

We go a step deeper and provide (in G_A_A) the body of the procedure for the addition of a scalar product to an accumulator:

```
procedure DOT_ADD(C   : in out DOT_PRECISION;
                  V,W : in     VECTOR_TYPE) is
    SHIFT : constant INTEGER := W'FIRST-V'FIRST;
begin
    if (V'LENGTH /= W'LENGTH) then
        raise INDEX_ERROR;
    else
        for I in V'RANGE loop
                        ADD_PROD(C, V(I), W(SHIFT+I) );
        end loop;
    end if;
end DOT_ADD;
```

A possible way of implementing ADD_PROD when a satisfactory hardware implemented DOUBLE_REAL is available is as follows:

```
procedure ADD_PROD( C : in out DOT_PRECISION;
                    S,T  : in   FLOAT_TYPE ) is
begin
   ADD( C, S * T )
   --       ! --double precision result of single precision arguments
end ADD_PROD;
```

In the absence of a hardware implemented DOUBLE_REAL, the conversion to and from the artificial intermediate storage is too wasteful. Therefore, we prefer the following and more general implementation:

```
procedure ADD_PROD( C : in out DOT_PRECISION;
                    S,T  : in   FLOAT_TYPE ) is
   S_INT,T_INT : INTERNAL_REAL;
begin
   if S/=0.0 and T/=0.0 then
      CONVERT(S, S_INT);
      CONVERT(T, T_INT);
      INTERNAL_ADD_PROD( C, S_INT, T_INT );
   end if;
end ADD_PROD;
```

This simplification has (together with the use of integer rather then real arithmetic) greatly contributed to the overall performance of the package.

INTERNAL_REAL is some private type, e.g. if a satisfactory floating-point DOUBLE_REAL is available, then the following declaration may do:

```
type INTERNAL_REAL is new DOUBLE_REAL;
```

More likely if INTERNAL_REAL is to be portably and efficiently implemented in Ada, and because we have chosen integer cells for the accumulator, then integers seem appropriate containers for the mantissa also. Therefore, we would recommend the following:

```
type INTERNAL_REAL is
   record
      EXP    : INTEGER;
      SGN    : BOOLEAN;
      MANTISA : LONG_VECTOR;
   end record;
   -- where --
   type LONG_VECTOR is array(1..ENVIRONMENT.LONG)of INTEGER;
```

although the separation of the sign is not strictly needed.

The name LONG has been used to indicate that some SHORT objects may be convenient in some cases for increased efficiency.

4.2. ROUNDINGS ON DOT_PRECISION OBJECTS

Since large exponent ranges imply greater accumulator lengths, a pair of pointers (START, FINISH) to the part of accumulators which is actually used has been provided. It saves us from initialising the whole of accumulator and examining its total length in the search for relevant information.

Three directed roundings are provided - UP, DOWN, and to the NEARest. Taking into account that the process to establish a particular rounding of ACCU.VALUE does not differ greatly from the other two roundings, it is most efficient to do all three at the same time. Also, an interval (DOWN,UP) enclosing the accumulator contents is often sought. Therefore it appears reasonable to declare ROUNDS as follows:

```
procedure ROUNDS( C : in out DOT_PRECISION;
                  DOWN,
                  UP,
                  NEAR : out FLOAT_TYPE);
```

By choosing the mode of C to be **in out** we allow implementations which change the contents of DOT_PRECISION objects but not the numerical value (e.g getting rid of leading and trailing zeros).

The actual implementation of ROUNDS is hindered by:

- the length of FLOAT_TYPE'MACHINE_MANTISSA
- the length of accumulator
- the need to carefully avoid underflow
- the need to search (sometimes much further) than FLOAT_TYPE'MANTISSA_LENGTH in order to determine downwardly and upwardly directed roundings.

It is relatively the most expensive operation and the obvious candidate for assembler when efficiency is critical; especially taking into account that later it will be used for scalar operations as well.

The above specification of the rounding process makes it very easy to implement 3 distinct and user friendly functions in the user interface package G_S_C: ROUND_NEAR, ROUND_UP, ROUND_DOWN.

As an example we demonstrate the case of rounding downwards:

```
function ROUND_DOWN(C:DOT_PRECISION) return FLOAT_TYPE is
   DOWN,UP,NEAR : FLOAT_TYPE;
   C_COPY       : DOT_PRECISION;
begin
   COPY  (C, C_COPY);
   ROUNDS(C_COPY, DOWN,UP,NEAR);
   return DOWN;
end ROUND_DOWN;
```

Because functions in Ada may only have **in** parameters, we must take an internal copy of the accumulator. To produce a copy of the **in** parameter C, we employ an (inlined) procedure COPY instead of assignment ':=' (although its body may be implemented as assignment) to enable DOT_PRECISION to be declared as a limited type. This is to allow an implementation of the accumulator in which the same numerical value may have different representations (e.g. one of them has trailing zeros). In order to do that predefined equality must be invalidated by means of a limited type declaration, for otherwise the wrong result may be returned.

Whether re-defined "=" will be made available to end-users depends on the functional requirements of G_S_C; but it must be visible to implementors of interval arithmetic in order to correctly deliver (at least) the body of interval scalar product; since it can happen that different extended-precision numbers have the same floating-point rounding.

We provide it in more general form:

```
type COMPARISON is (LESS,EQUAL,GREATER);
                        -- and then --
procedure COMPARE(C,Z : in out DOT_PRECISION;
                        RES :   out COMPARISON );
```

so that comparisons on floating-point numbers may be (accurately) redefined.

4.3. GENERIC_ENVIRONMENT_ENQUIRIES

The calculation of constants to define the accurate environment (e.g. the size of the accumulator) is straightforward but requires some complicated expressions to determine their values. Such expressions are not permitted in the declarative part of an Ada package; we therefore calculate such constants in a separate auxiliary package GENERIC_ENVIRONMENT_ENQUIRIES.

For efficiency we also precalculate some useful expressions derived from such constants (for example, pointers to the used part of an accumulator for each possible exponent value). This once and for all precalculation of global constants contributes to efficiency; although at the expense of large arrays when the exponent range is huge.

5. EFFICIENCY CONSIDERATIONS

Real scalar operations could all be implemented in terms of the scalar product as follows:

```
S * T = ( S )   * ( T )    -- vectors of length 1
S + T = ( S , T ) * ( 1 , 1 )  -- vectors of length 2
S - T = ( S , T ) * ( 1 ,-1 )  -- vectors of length 2
```

Apart from division (which is usually the most tricky part anyway), it seems that (at least in principle), there are no problems with implementing accurate scalar operations.

There is an immediate attraction to the above approach. The efficiency of the whole package depends on a single subprogram, which could be written in assembler or provided in hardware if Ada code should prove inadequate for certain applications. The cut between Ada and hardware/assembler has never been so simple!

However, in order to provide an efficient but portable product, we will slightly deviate from this simple concept.

The obvious move would be to do a single conversion of 1.0 (and −1.0) to the INTERNAL_REAL representation, once and for all.

Secondly, if the software INTERNAL_REAL is very compatible with the accumulator structure (and it should be) to allow an efficient INTERNAL_ADD_PROD; then it may happen that it is not too friendly

with respect to addition/subtraction. Simply speaking, multiplication requires that the mantissa of a factor is more thinly spread through several integers so that the partial products do not overflow. That is not the case with addition, and we may very well use an extra auxiliary type:

```
type DENSELY_PACKED is
   record
      EXP    : INTEGER;
      SGN    : BOOLEAN;
      MANTISA : SHORT_VECTOR;
   end record;
   -- where --
   type SHORT_VECTOR is array(1..ENVIRONMENT.SHORT)
      of INTEGER;
```

Not only is the declaration almost identical to that of INTERNAL_REAL, but so is the conversion CONVERT_SHORT to this SHORTer internal representation of floating-point numbers.

Thirdly, in order to calculate:

```
r( S & T)        -- & stands for any arithmetic operation
```

for the rounding r and scalars S,T it is sufficient to calculate an approximation S_T such that:

```
r( S & T) = r( S_T )
```

Therefore, when adding (subtracting) a relatively small T to a larger S we may actually add only the relevant part of T'MANTISSA; or just mark that there is some positive/negative tail behind S'MANTISSA.

5.1. INTERNAL TYPES

A close look at complex multiplication or division will reveal that there are floating-point objects which are converted to INTERNAL_REAL more then once:

```
function "/" (S, T : COMPLEX_TYPE) return COMPLEX_TYPE is
   RE_PART, IM_PART : DOUBLE_REAL;
   R2 : DOUBLE_REAL;
begin
   R2:= T.RE * T.RE + T.IM * T.IM;
   RE_PART := ( S.RE * T.RE + S.IM * T.IM ) / R2;
   IM_PART := ( S.IM * T.RE - S.RE * T.IM ) / R2;
   return (ROUND_NEAR (RE_PART), ROUND_NEAR (IM_PART));
end "/";
```

In this pseudo-Ada notation for the division algorithm both arguments T.RE, T.IM will be converted 4 times instead of just once, while each of S.RE, S.IM will be handled twice!

An even more interesting situation occurs when doing matrix*matrix and matrix*vector multiplication. Consider the case of a software-implemented scalar product used naively to implement a matrix multiplication A*B: vector components are bound to be converted to some internal representation. Alas! Each component of matrix A will be converted B'LENGTH(2) times instead of once. Each component of matrix B will be converted A'LENGTH(1) times instead of once. The possible saving in the

case of A'LENGTH(1)=A'LENGTH(2)= B'LENGTH(2)=100 is 2*100**2*(100–1) = 1 980 000 conversions!

Since conversion is certainly not the only operation, we conservatively estimate the increase in efficiency is up to 40% for any sizable array'LENGTH. In practice, it can be even better.

The above idea for minimising conversions requires only a storage-conscious algorithm that does not convert both matrices at the same time. For example to multiply two matrices A and B, we can convert B column by column and store in internal workspace; the matrix A can be converted a row at a time when needed.

More modestly, one could convert only the rows of A, one at a time, but not the columns of B.

Even in the presence of above facilities, situations like interval defect iteration:

```
for i in 1..N loop
   X := Z + B*(E*X);
end loop;
```

where E is a constant INTERVAL, X and Z are INTERVAL_VECTORs and B is a constant INTERVAL_MATRIX, deserve some attention. Otherwise, it may happen that B (and E) will be converted to their internal representation N times.

We can explicitly introduce internal types which define internal representations for vectors and matrices:

```
type INTERNAL_VECTOR is
   array(INTEGER range <>)of INTERNAL_REAL;
INTERNAL_MATRIX is
   array(INTEGER range <>, INTEGER range <>)of INTERNAL_REAL;
```

and can provide operations for such internal vector and matrix types. With such internal types it is then possible to write higher level subprograms using the internal types explicitly within their bodies, thereby minimising the number of conversions to internal form.

For example in the solution of a linear system of equations Ax = b, the access to operations on the internal representation reduces the number of conversions from N*(N+1)*(8*N–7)/12 + N**2 to N**2 + 2*N – 1, where N is the order of the system.

6. CONCLUSION

We have demonstrated that high precision accurate arithmetic as designed by *Kulisch & Miranker* can be implemented in a portable yet efficient manner using the Ada language.

Usable, readable modules are produced that provide the operations belonging to this design related to the different number spaces for which the accurate arithmetic is valid. This allows the use of these facilities for large scientific applications including accuracy-critical calculations. In this task the Ada language appears to be a useful tool.

Care has been taken that the designed facilities are compatible with other available general Ada software, so users can take benefit of several Ada utilities without loss of efficiency.

The project results accommodate both regular users who require the safe, modular and user-friendly environment provided by the package GENERIC_SCIENTIFIC_COMPUTATION, and expert users who

require the flexibility to extend and tailor (to their specialist needs) the efficient primitive operations (whose implementation is private) provided by the package GENERIC_ACCURATE_ARITHMETIC.

Finally, the implementation can be tailored to the particular hardware environment in an automatic way without the user needing to be aware of the private details of the implementation. Indeed by ensuring the user cannot use low level details of the implementation, we can use such features in an efficient and sophisticated way in the safe knowledge that such features cannot be abused by a naive user. No matter what implementation we choose at the lower level (closest to the machine level), the user specification remains the same.

REFERENCES

[1] ANSI/IEEE Std 754-1985. *IEEE Standard for Binary Floating-Point Arithmetic*, July 1985.

[2] ANSI/MIL-STD 1815 A. *Reference manual for the Ada programming language*, January 1983.

[3] Bohlender, G., Rall, L.B., Ullrich, C., and Wolff von Gudenberg, J. *Pascal-SC*, Bibliographisches Institut, Mannheim, 1986.

[4] DIAMOND Project *Development and Integration of Accurate Mathematical Operations in Numerical Data processing*, ESPRIT Project 1072.

[5] Klatte, R., Ullrich, C.P., and Wolff von Gudenberg, J. Arithmetic specification for scientific computation in Ada, *IEEE Transactions on Computers*, Vol. c-34.11, Nov 1985, 996-1005.

[6] Kulisch, U.W. and Miranker, W.L. *Computer arithmetic in theory and practice*, Academic Press, 1981.

[7] Kulisch, U.W. and Miranker, W.L. (eds.) *A new approach to scientific computation*, Proceedings of the "IBM Symposium" in August 1982, Academic Press, 1983.

[8] Kulisch, U.W. and Miranker, W.L. (eds.) The arithmetic of the digital computer: a new approach, *SIAM Review*, Vol.28.1, March 1986.

Project No. 1072

A PROCEDURE FOR THE EVALUATION OF ARITHMETIC EXPRESSIONS WITH GUARANTEED HIGH ACCURACY*

Authors:

H.C. Fischer
Universität Karlsruhe, Institut für Angewandte Mathematik
Kaiserstraße 12
7500 Karlsruhe, W. Germany

R. Haggenmüller
Siemens AG München, Geschäftsbereich Datentechnik
Otto-Hahn-Ring 6
8000 München 83, W. Germany

G. Schumacher
Universität Karlsruhe, Institut für Angewandte Mathematik
Kaiserstraße 12
7500 Karlsruhe, W. Germany

The mathematical theory of computer arithmetic developed by U. Kulisch and W. Miranker provides a sound base for solving many numerical problems with guaranteed high accuracy. In this paper it is used to develop a procedure for the evaluation of arithmetic expressions. The expressions are transformed into systems of nonlinear equations which are solved by a residue correction procedure. During the solution process the quality of the computed result is controlled with help of enclosing intervals.

0. Introduction

In the FORTRAN-libraries ARITHMOS (Siemens AG) [10] and ACRITH (IBM) [9] routines are contained providing guaranteed bounds for the value of an arithmetic expression defined by +, −, •, / and powers with constant integer exponents.

Within the framework of the ESPRIT-project DIAMOND [4] the authors developed a formula evaluation program in the PASCAL extension PASCAL-SC [2], which has a functionality enlarged by the following points:

1. variable integer exponents;
2. evaluation of expressions containing standard functions;
3. treatment of tolerance-afflicted data, i.e. input of interval values for the variables.

Before going into technical details, we briefly like to outline what kind of problems we are confronted with and which instruments we use to solve them.

---

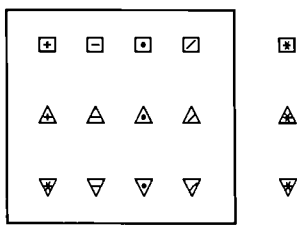
* This paper is part of work in the ESPRIT-project DIAMOND, project no. 1072.

The floating-point arithmetic, usually used in scientific computation, confronts us with an apparently paradox situation. On the one hand, most of the computers available on the market today do floating-point basic operations with high accuracy, on the other hand, however, results from scientific computations may differ seriously from the actual value. Two examples may illustrate this:

> The expression $x^{30} + 777 - x^{30}$, evaluated at the place x = 10, has the value 777. Nevertheless almost all floating-point-computers give as result the value 0.
>
> The evaluation of the expression $9x^4 - y^4 + 2y^2$ for the values x = 10864 and y = 18817 gives, on a computer with a 13-digit-decimal arithmetic, the value 58978. But the true value is 1 [8].

The appearing effects (rounding errors and their propagation) arise at the transition of the real numbers $\mathbb{R}$ (in which the formula is defined) to their subset S, the machine numbers. To get the situation under control, we use a computer arithmetic, which was developed by a group of mathematicians under the direction of Kulisch und Miranker [6].

This theory is essentially based on the so-called 15 basic operations:



These are the four basic operations +, −, ·, / and the scalar product $*$, each with three different roundings $\square$, $\triangle$, ∇ (near, up, down). It is

$$a \bigcirc b := \bigcirc(a \circ b),$$
$$\forall\, a,b \in S,\ \forall \circ \in \{+,-,\cdot,/\},\ \forall \bigcirc \in \{\square,\triangle,\nabla\},$$

i.e. the result on the machine is defined as the rounded value of the exact result. This seemingly trivial claim has been achieved so far only on few computers (in hardware). The new IEEE-standard for floating-point operations at least guarantees the bordered part of the basic operations. The scalar product operations, however, are absolutely necessary for matrix-vector operations, as well as the accuracy improvement methods for numeric procedures still dealt with in the following. Therefore, in addition, for two vectors a and b over S it is:

$$a \circledast b := \bigcirc(a * b)\ , \quad \forall \bigcirc \in \{\square, \triangle, \nabla\}\ . \quad (1)$$

The described computer arithmetic on its own, however, is not sufficient to get fully rid of the above-mentioned difficulties. It makes it possible to do interval computation [1] and offers the possibility to compute bounds for the solution, but if the diameter of the interval for the solution is too large, the result may be worthless. The described arithmetic, however, can be applied in such a way that even complicated problems can be solved with high accuracy. We will prove this in the case of the formula evaluation program.

1. The transition from an arithmetic expression to a system of nonlinear equations

The transition from an arithmetic expression to a system of nonlinear equations shall be demonstrated by the following example.
Let f be the expression

$$f = (a + b)^n \cdot (c - d) / e \ .$$

Usually the following intermediate results occur with the evaluation of this expression:

$$\begin{aligned} z_1 &:= a + b \\ z_2 &:= z_1^n \\ z_3 &:= c - d \\ z_4 &:= z_2 \cdot z_3 \\ z_5 &:= z_4 / e \ . \end{aligned}$$

with $f = z_5$. Hence the computation of f corresponds with the solution of the system of nonlinear equations

$$\begin{aligned} z_1 - a - b &= 0 \\ z_2 - z_1^n &= 0 \\ z_3 - c + d &= 0 \\ z_4 - z_2 \cdot z_3 &= 0 \\ e \cdot z_5 - z_4 &= 0. \end{aligned}$$

The general case is treated analogously: The expression is transformed into the postfix form and for every operator an intermediate result is introduced.

2. Solving special systems of equations

Now we have to deal with the solution of systems of equations of the following kind:

$$\begin{aligned} g_1(z_1) &= 0 \\ g_2(z_1, z_2) &= 0 \\ &\vdots \\ g_n(z_1, \ldots, z_n) &= 0 \end{aligned} \qquad (2)$$

where $g_i : D_1 \times \cdots \times D_i \to \mathbb{R}$, $D_j \subset \mathbb{R}$, $i=1,\ldots,n$, $j=1,\ldots,i$. Shortening, we also write $D := D_1 \times \cdots \times D_n$. The quantities z_i correspond in our previous presentation with the intermediate results appearing within the evaluation of a formula. The whole system is to be solved by successive forward-solving which exactly amounts the "normal" evaluation of a formula. Furthermore in our context the partial derivatives with respect to z_j exist for all g_i. If now

$\tilde{z}_1,\ldots,\tilde{z}_n$ are computed approximations (possibly differing extremely from the actual solution $\hat{z}_1,\ldots,\hat{z}_n$), then in the following a possibility is described how to get inclusions for the defects

$$\Delta z_i := \hat{z}_i - \tilde{z}_i \ , \ i=1,\ldots,n \qquad (3)$$

We use the following abbreviating notation:

For $x,y \in D$, g_k from (2), $i \in \{1,\ldots,k\}$ and $\zeta \in x_i \cup y_i$
we define

$$\begin{aligned} r_i(g_k,x,y) &:= g_k(x_1,\ldots,x_{i-1}, y_i,\ldots,y_k) \\ s_i(g_k,x,y,\zeta) &:= \frac{\partial}{\partial z_i} g_k(x_1,\ldots,x_{i-1},\zeta,y_{i+1},\ldots,y_k) \end{aligned} \qquad (4)$$

Instead of $r_1(g_k,x,y)$ we write briefly $r(g_k,y)$. If $\frac{\partial}{\partial z_k} g_k$ is independent of z_k, we replace $s_k(g_k,x,y,\zeta)$ by $s_k(g_k,x)$.
We now want to expand the k-th equation of (2) according to the mean value theorem at point $\tilde{z} = (\tilde{z}_1,\ldots,\tilde{z}_n)^T$; with (3) and (4) we get in a first step

$$g_k(\hat{z}_1,\ldots,\hat{z}_k) = r_k(g_k,\hat{z},\tilde{z}) + s_k(g_k,\hat{z},\tilde{z},\zeta_k)\cdot\Delta z_k$$

and finally after k steps

$$g_k(\hat{z}_1,\ldots,\hat{z}_k) = r(g_k,\tilde{z}) + \sum_{j=1}^{k} s_j(g_k,\hat{z},\tilde{z},\zeta_j)\cdot\Delta z_j .$$

Since $\hat{z}_1,\ldots,\hat{z}_k$ is the exact solution of (2), the left side of this expression disappears. For Δz_k we get the formula

$$\Delta z_k = \Big(-r(g_k,\tilde{z}) - \sum_{j=1}^{k-1} s_j(g_k,\hat{z},\tilde{z},\zeta_j)\cdot\Delta z_j \Big) \ / \ s_k(g_k,\hat{z},\tilde{z},\zeta_k) \qquad (5)$$

provided that

$$s_k(g_k,\hat{z},\tilde{z},\zeta_k) \neq 0.$$

For k = 1 (5) looks like this:

$$\Delta z_1 = -r(g_1,\tilde{z}) \ / \ s_1(g_1,\hat{z},\tilde{z},\zeta_1) \qquad (6)$$

If $s_1(g_1,\hat{z},\tilde{z},\zeta_1) = \frac{\partial}{\partial z_1} g_1(\zeta_1)$ is independent of z_1, then (6) is an appropriate formula for the computation of an inclusion $[\Delta z_1]$ of Δz_1.

By means of induction it is easy to show that with inclusions $[\Delta z_1],\ldots,[\Delta z_{k-1}]$ for $\Delta z_1,\ldots,\Delta z_{k-1}$ the inclusion $[\Delta z_k]$ of Δz_k can be computed with formula (5).

$s_k(g_k,\hat{z},\tilde{z},\zeta_k)$ has to be independent of z_k (i.e. g_k is at most linear in z_k). For each ζ_i the whole interval $\tilde{z}_i \cup \hat{z}_i$ is substituted; in doing so we take advantage of the already existing information

$$\hat{z}_i \in \tilde{z}_i + [\Delta z_i] \ , \quad i=1,\ldots,k-1.$$

We thus obtain the inclusion formula

$$\Delta z_k \in [\Delta z_k] := \Big[-r(g_k,\tilde{z}) - \sum_{j=1}^{k-1} s_j(g_k,\ \tilde{z}+[\Delta z],\ \tilde{z},\ \tilde{z}_j \cup (\tilde{z}_j+[\Delta z_j]))\cdot[\Delta z_j]\Big] \ / \ s_k(g_k,\ \tilde{z}+[\Delta z]) \qquad (7)$$

The quality of an inclusion computed according to this formula heavily depends on the accurate calculation of the residue term $r(g_k,\tilde{z})$. The necessary accuracy can be easily achieved with help of the scalar product.
The assumption that $\frac{\partial}{\partial z_k} g_k$ does not depend on z_k is always fullfilled for that kind of equations which occur with appropriate transformation of a formula into a system of equations.

In all concrete cases occurring within the evaluation of arithmetic expressions, formula (7) is in comparison to the general case of a much easier shape. Since the formulae can be easily derived form (7) we only give one example:

Addition

$$g_k\ (z_1,\ldots,z_k) = z_k - (z_i + z_j) \ , \quad i < j < k$$
$$[\Delta z_k] = -\tilde{z}_k + \tilde{z}_i + \tilde{z}_j + [\Delta z_i] + [\Delta z_j]$$

3. An algorithm for the evaluation of an arithmetic expression

As indicated in chapter 2, the idea of the computation of bounds for the value of a formula is essentially based on two steps:

(a) Computation of an approximation for all intermediate results $\tilde{z}_k$ (k=1,...,n);
(b) Computation of inclusions $[\Delta z_k]$ for the defects Δz_k (k=1,...,n).

If the quality of the result $\tilde{z}_n + [\Delta z_n]$ is not sufficient the midpoint $m([\Delta z_k])$ of $[\Delta z_k]$ (k=1,...,n) will be used to improve the approximation $\tilde{z}_k$ to

$$\tilde{z}_k + m([\Delta z_k]). \qquad (8)$$

Then one starts again with (b).
The improvement of the approximation, however, is not done by adding $\tilde{z}_k$ and

$m([\Delta z_k])$ explicitely, but by storing $\tilde{z}_k$ and $m([\Delta z_k])$ in a correction vector. After the first step we set: $z_k^{(0)} := \tilde{z}_k$, $z_k^{(1)} := m([\Delta z_k])$ and $[\Delta z_k^{(1)}] := [\Delta z_k]$. In the r-th correction step the approximation $\tilde{z}_k$ now looks like

$$\tilde{z}_k = \sum_{s=0}^{r} z_k^{(s)}.$$

Of course this has to be taken into account in the formula of chapter 2. The formula for the computation of the r-th correction, for example as far as the multiplication of two intermediate results is concerned, can be read like this:

$$[z_k^{(r)}] := \diamond\left[-\sum_{s=0}^{r-1} z_k^{(s)} + \sum_{s=0}^{r-1}\sum_{t=0}^{r-1} z_i^{(s)} z_j^{(t)}\right] \diamondsuit \diamond\left(\sum_{s=0}^{r-1} z_j^{(s)}\right) \diamondsuit [z_i^{(r)}] \diamondsuit$$
$$\left[\diamond\left(\sum_{s=0}^{r-1} z_i^{(s)}\right) \diamondsuit [\Delta z_i^{(r)}]\right] \diamondsuit [\Delta z_j^{(r)}].$$

By $\diamond r$ ($r \in \mathbb{R}$) we denote the smallest interval with bounds from S, which contains r, by $\diamondsuit$ and $\diamondsuit$ the interval addition and multiplication.
Obviously the residue term may be interpreted as a scalar product, i.e. as a sum of products. Therefore we can use the scalar product (1). This is also possible in the other cases.
For standard functions and interval parameters a slightly different approach is necessary. The interested reader is refered to an article in SIEMENS Reports on Research and Development [5]. There the treatment of interval input by a subdivision method [7] is described in some detail.

The algorithm looks like this:

1. Compute approximations $z_k^{(0)}$ of all intermediate results (k=1,..,n);

2. Iteration
   r := 0;
   repeat
     r := r+1;
     if r > 1 then $z_k^{(r-1)} := m([z_k^{(r-1)}])$ (k=1,...,n);
     Compute inclusions $[z_k^{(r)}]$ (k=1,...,n) according to the formula from chapter 2;
     $[y^{(r)}] := \diamond\left(\sum_{j=0}^{r-1} z_n^{(j)}\right) \diamondsuit [z_n^{(r)}]$; {formula value}
   until (d($[z_n^{(r)}]$) $\leq \delta$ | $[y^{(r)}]$ |)
   {d(A) denotes the diameter of interval A}
     or ($[y^{(r)}] = [y^{(r-1)}]$)
     or (underflow and r $\geq$ 10);

Algorithm 1

The first termination criterion applies to the case that the desired accuracy has been achieved. Proceeding on the assumption that $\delta=B^{-t}$, with B being the basis for the number presentation and t the mantissa of the machine, a result is expected whose upper and lower bounds differ only in the last digit [3].
The second criterion is used for small-tolerances-afflicted input data. It may occur that the diameters of these intervals do no longer render possible a further improvement of the result.
At last, the third criterion is necessary if during computation an underflow takes place. This may indicate that an intermediate result needs more digits than those given by the actual floating-point numbers. If this is the case the termination should not take place until 10 corrections have been tried. 10 is a useful value for a domain of exponents reaching from -99 to +99.

With a view to briefing, we do not go into further details such as overflow handling or avoiding divisions by zero. It may be enough, to mention here that these exceptions are dealt with by recursive application of the formula evaluation procedure to the critical argument.

4. Concluding Remarks

The requirement of getting models which approximate reality in a better way has as consequence the tying up of complicated formulae. Symbolic manipulation even makes it possible to create formulae which cannot be mastered manually. But how is it possible to handle such formulae if we have no reliable procedures at hand which relieve the scientist or engineer from control of the computed results? There is no doubt that the introduced formula evaluation program is such a procedure. Unlike other instruments with similar characteristics - e.g. symbolic manipulation - our introduced procedure tries to obtain the desired information by means of a correction technique on a floating-point basis with an expenditure that is as minimal as possible. Besides, the applied technique can be enlarged naturally to expressions of another kind, e.g. matrix-vector expressions.

An operational area for a formula evaluation program that may not be neglected is of course the solution of systems of nonlinear equations. Just near solutions (zeros) it is unavoidable that certain cancellation effects occur. Problems similar to those mentioned at the beginning of this paper in the trivial examples are inevitable and cannot be understood as easily as it could be done there.

References

[1] Alefeld, G. und Herzberger, J.: Einführung in die Intervallrechnung, Bibliographisches Institut, Mannheim, 1970

[2] Bohlender, G., Rall, L.B., Ullrich, Ch. und Wolff von Gudenberg, J.: PASCAL-SC, Bibliographisches Institut, Mannheim, 1986

[3] Böhm, H.: Berechnung von Polynomnullstellen und Auswertung arithmetischer Ausdrücke mit garantierter maximaler Genauigkeit, Dissertation, Universität Karlsruhe, 1983

[4] Fischer, H.C., Haggenmüller, R., Schumacher, G.: Evaluation of Arithmetic Expressions, DIAMOND, Deliverable D2a-1, Doc. No.: 03/2a-1/1/K02.f

[5] Fischer, H.C., Haggenmüller, R., Schumacher, G.: Evaluation of Arithmetic Expressions with Guaranteed High Accuracy, to appear in Siemens Forschungs- und Entwicklungsberichte, September 1987

[6] Kulisch, U. und Miranker, W.L.: Computer arithmetic in theory and practice, Academic Press, New York, 1981

[7] Ratschek, H. und Rokne, J.: Computer methods for the range of functions, Ellis Horwood, Chichester, 1984

[8] Rump, S.M.: How reliable are results of computers, Jahrbuch Überblicke Mathematik 1983, Bibliographisches Institut, Mannheim, pp. 163-168

[9] ACRITH: IBM High-Accuracy Arithmetic Subroutine Library: Program Description and User's Guide, 1986

[10] ARITHMOS (BS 2000): Benutzerhandbuch, SIEMENS Softwareprodukt V1.0A, 1986

Project No. 410

ESTELLE AND LOTOS SOFTWARE ENVIRONMENTS FOR THE DESIGN OF OPEN DISTRIBUTED SYSTEMS

Michel Diaz *, *Chris Vissers* **, *Stanislaw Budkowski* ***

* LAAS du CNRS
7, avenue du Colonel Roche
31077 TOULOUSE CEDEX
France

** University of TWENTE
Dept. Informatics
7500 AE ENSCHEDE
The Netherlands

*** BULL - DSAS
68, route de Versailles
78430 LOUVECIENNES
France

ABSTRACT : The objectives of the ESPRIT-SEDOS ST 410 project are to assess, define and develop formal techniques and related tools for the design of hierarchies of software in complex distributed systems. The selected approach is based on the development of two formal techniques, ESTELLE and LOTOS, which are being developed within ISO, because of the resulting impacts and interests. Shortly after the ESPRIT Technical Week of 1987 the SEDOS project will come to an end. This paper describes the main results of the project in the light of its original objectives.

I. INTRODUCTION

The implementation of complex computer and local networks implies to design sophisticated communicating software. Defining and realizing the needed protocols proves to be quite difficult because the designer is, in the general case, faced to a rather sophisticated hierarchy of software layers and it follows that using formal description techniques provides a good and strong support for managing the corresponding complexity.

Formal Description Techniques have also been recognized of importance and adopted by the OSI environment as being indispensable for unambiguous, concise, and clear specification of the generally complex functions defined in services and protocols. Furthermore, Formal Description Techniques provide the basis for analysing specifications with respect to their correctness, completeness and consistency, and for checking the conformance of implementations with respect to specifications [27].

A short description of the SEDOS Project (Software Environment for the Design of Open distributed Systems) will be given first, before describing its main results.

II. OVERVIEW OF SEDOS

The SEDOS Project has been dedicated to the support of the Formal Description Techniques (FDTs) ESTELLE and LOTOS that are developed within ISO for the description of OSI protocols and services.

OSI standards for protocols and services are characterized by two major factors: their substantial functional complexity, and the requirement that they

have to be specified independently of any implementation constraints (in order to allow the freedom of all possible implementations). These characteristics impose unprecedented requirements to the definition of FDTs with respect to power of expression and level of abstraction.

SEDOS aimed to assess, define, and partially develop the ESTELLE and LOTOS formal techniques and related software support tools for the design of services and protocols in distributed architectures. The architectures are supposed to be in conformance with the OSI layering principle, as this is recognized as the only way to successfully realise complex distributed systems.
The global objectives of the project were :
- to advance the definition of the two ISO FDTs, ESTELLE and LOTOS, to a level where the above stated requirements can be guaranteed; this includes contributions and participations to ISO and national member bodies;
- to demonstrate their adequateness by providing complete formal descriptions of real and complex OSI protocols and services, and
- to start with the development of software tools that aim at the support of the protocol design cycle from specification to implementation.

a) *Modelling*
Many description models have been developed, more or less formal and close to protocol reality, in particular based on: extended state machines [6] and Petri nets [13-18], dedicated languages [3-19], ISO Formal Description Techniques TC97 SC21 WG1 FDT: ESTELLE [20], LOTOS [21]. ESTELLE is based on extended state machines communicating through FIFOs and LOTOS is based on temporal ordering of events.
In SEDOS, ESTELLE and LOTOS have been selected because of their interests: their main specificities are given in [15] and their formal syntax and semantics are described in ISO DIS 9074 [20] and DIS 8807 [21].

b) *Tools*
It appears that formal description techniques are of the utmost importance when dealing with complex systems; for design efficiency, it appears quite necessary to derive from FDTs support tools.
These tools are meant to help the designer efficiently in:
- formally expressing the specification of protocols and services in distributed architectures,
- verifying services and protocols by formal methods,
- implementing specifications by means of an FDT simulator and compiler,
- deriving preliminary test scenarios from specifications.

c) *SEDOS*
Such a set of techniques and tools has been investigated in SEDOS through the cooperation of eleven organizations of six countries (LAAS du CNRS, University of TWENTE, ICL, BULL, ADI, University of CATANIA, Politecnico of MILANO, INRIA, HMI, Politecnico of MADRID, Technical University of BERLIN). Other projects are being developed, but their scopes are much narrow than SEDOS one. Started in November 1984, for 3 years, the general description of the SEDOS objectives, interests and overall organization appears in [14].

c) *General comments*
At the end of the Project, it can be observed that the global objectives will be achieved:

i) ESTELLE and LOTOS are now in the stage of Draft International Standards. International Standards are expected around the middle of 1988. Without hesitation it can be stated that the contribution of SEDOS in this development has been paramount.
ii) The progression of ESTELLE and LOTOS has been accompanied by extensive trial specifications of OSI protocols and services, in particular for the standards of the Transport and Session layers.
iii) A set of prototype tools have been developed for verification and simulation. The simulators for ESTELLE and LOTOS will be demonstrated during the ESPRIT week.

It may be emphasized that the trial specifications served several purposes of which two are worth to be mentioned here:
First, the expressive power and abstraction level of an FDT can only be adequately assessed on basis of real world experience, i.e. experience with the complete description of complex OSI standards. This work indeed has led to significant improvements to the definition of the FDTs as will be explained in the separate sections about ESTELLE and LOTOS.
Second, it has been demonstrated to the target user community, which generally is unfamiliar with FDTs and not per definition in favour of its application, what effects can be expected from the application of FDTs.

In this respect it can be observed that within the ISO Member Bodies there is not yet enough knowledge and expertise to review a formal description of a standard and confirm that such a description faithfully reflects the standard. For that reason ISO proposes a progressive introduction of the application of FDT to their standards in three phases. In the first phase the natural language description of the standard will prevail as the authoritative description of the standard and the formal description will be published separately as ISO Technical Report type 2. In the second phase the formal description will be attached to the standard as a non binding annex. Only in the third phase, when enough knowledge and expertise is available in the Member Bodies the formal description can replace the natural language description as the authoritative description. Meanwhile ISO will work on the development of educational material in order to spread the knoweldge and experience with FDTs in the Member Bodies.

Whereas the slow penetration of FDTs into the OSI area may seem rather discouraging, it also gives a clear picture of the advanced position of SEDOS and Europe in the development and application of FDTs.

The development of software tools for the support of the protocol design cycle proved to be a difficult but challenging effort. The fact that the definition of the FDTs were also under development while the tools were developed did not make this job particularly easier.

For each of the FDTs a set of prototype tools will become available but the selected approaches in both techniques differ in major respects due to the different models underlying these two FDTs. The ESTELLE technique is less abstract in nature and the tools naturally aim more directly at the implementation phase. The LOTOS technique is more abstract in nature and the first set of tools naturally aim at the architectural specification phase. This difference and its consequences will appear in what follows.

The two following sections present the main results which have been achieved for ESTELLE and LOTOS and gives a particular attention to specifications and tools.

III. ESTELLE

ESTELLE is one of the description techniques which all are to serve as means to remove ambiguities from ISO protocol standards in which a combination of a natural language, state tables, etc. has been traditionally used. But an unambiguous formal specification still may be far from any implementation. There is a vital need for specifications of distributed systems in general, and communication protocols in particular which, being unambiguous and formalized, would at the same time indicate how implementations may be derived from them. We are convinced that this is precisely where the principal field of application for ESTELLE is situated. The semantics for ESTELLE have been formally defined, justifying the claim that ESTELLE is a Formal Description Technique.

ESTELLE has benefited from experiments in using previous description techniques and in with CCITT which defined SDL (Specification and Description Language) with which ESTELLE has some notions in common.

III.1. *ESTELLE definition and principal features*

A distributed system specified in ESTELLE is viewed as a collection of communicating components called in this paper tasks. Each task has a number of input/output access points called interaction points. A task will be represented graphically as an rectangle with points on its boundary.

The internal behavior of a task is described in terms of a nondeterministic communicating state automaton (a transition system) whose transition actions are given in the form of Pascal statements.

The tasks may be nested. This hierarchical (tree) structure may be depicted as in (Figure 1.a.) or as in (Figure 1.b.). The parent/children relationship is represented by edges or nested boxes. The root of the tree (the enclosing box) is the main task representing the specified system.

Besides the hierarchical task structure a communication structure exists within a specified system. The elements of this structure can be represented (Figure 2) by line segments which bind tasks' interaction points.

Two communication mechanisms are used in ESTELLE in order to enable cooperation between tasks:
- message exchange,
- restricted way of sharing variables.

The tasks may exchange messages, called interactions. A task can send an interaction to another task through a previously established communication link between two interaction points of the tasks. An interaction received by a task at its interaction point is appended to an unbounded FIFO queue associated to this interaction point.

Some variables can be shared between a task and its parent task. These variables have to be declared as exported by the children task. This is the only way variables may be shared. The simultaneous access to these variables is excluded because the execution of the parent's actions has always priority (so called parent/chrildren priority principle of ESTELLE).

Two kinds of parallelism between tasks can be expressed in ESTELLE:
- asynchronous parallelism,
- synchronous parallelism.

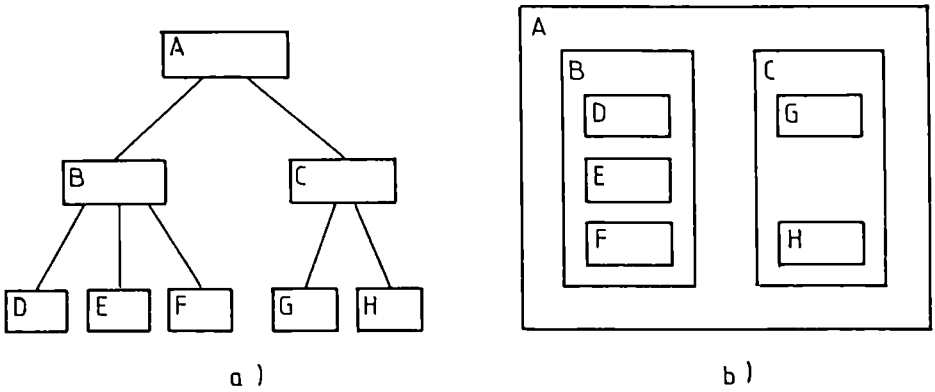


FIGURE 1

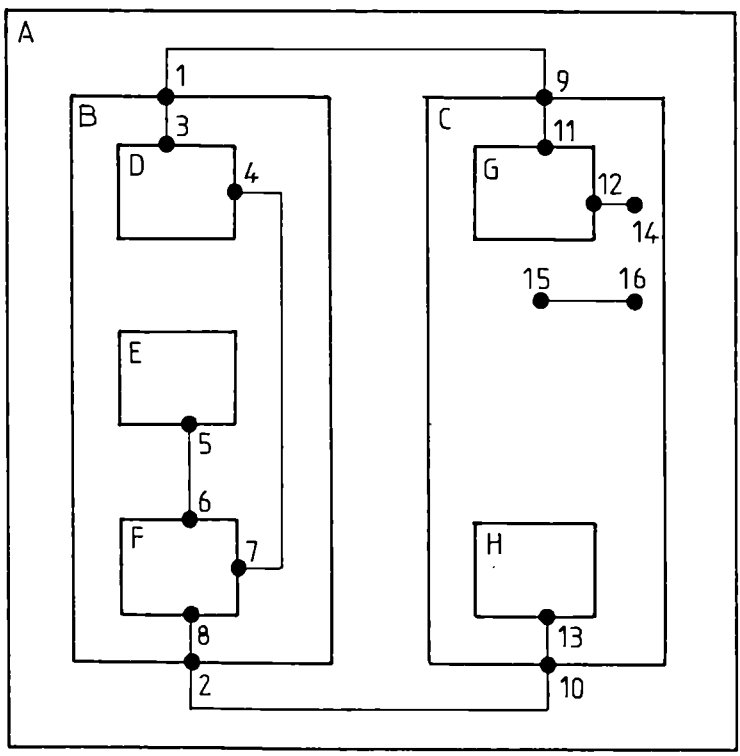


FIGURE 2

The asynchronous parallelism is authorized only between subsystems, or more precisely, between actions of different tasks belonging to the different subsystems. The synchronous parallelism is authorized only within a subsystem, or more precisely, between actions of different tasks belonging to the same subsystem.

Intuitively speaking, each subsystem runs by its own "computation steps". A computation step begins by a selection of one or a set of transitions (actions) among those "ready-to-fire", or "offered-to-fire" by the subsystem's tasks (at most one transition per task). Then these transitions are executed (in parallel) and, when all of them completed, the next computation step begins. That way the relative speed of tasks within a subsystem may be, in general, controlled (synchronized). That is why it is said that the parallelism within a subsystem has a synchronous character.

The subsystems, although they may exchange messages, run asynchronously in that their computation steps are completely independent from each other. The relative speed of evolution of subsystems are not at all controlled (synchronized).

In order to describe the behavior of the complete system specified in ESTELLE, the operational style (operational semantics) has been chosen.

The operational ESTELLE semantics considerably aids in specifying associated tools such as a simulator, a debugger and a compiler. It also reenforce a belief of the mutual compatibility of these tools, even if they are designed by different teams.

The global behavior of a system specified in ESTELLE is defined by the set of all possible sequences of, so called, "global situations" generated from the initial situation. Two consecutive global situations correspond to the execution of a transition, i.e., transitions are consider atomic in that, conceptually, one cannot observe intermediate results of their execution.

The ESTELLE semantics describe the way these sequences are generated, i.e., the way the system's transitions (transitions of its tasks) may be interleaved to properly model the synchronous parallelism within subsystems combined with the asynchronous parallelism between them.

The notion of time does not appear explicitly in ESTELLE. This is a consequence of the retained hypothesis that execution times of actions is not known. This knowledge is considered implementation dependent. However, to interpret properly "delays" (i.e., dynamic values assigned to some transitions which indicate a number of time units execution of these transitions must be delayed) a independent time process is assumed to exist. The computation model of ESTELLE outlined above is dependent on this time process only in that a relationship between progress of time and computation is defined and the "delay-timers" are observed in order to decide whether a transition can or can not be fired.

The formulated constraints specify a class of acceptable time processes. In each implementation or for simulation purpose any element of this class may be chosen.

III.2. *ESTELLE specifications*

Several ESTELLE descriptions of OSI protocols and services have been produced within the SEDOS Project. The following descriptions have a good

level of maturity as they reached a quite stable status and were parsed by the ESTELLE compiler developed within SEDOS:
- description of the ISO Network service;
- description of the ISO Transport service;
- description of classes 0 and 2 of the ISO Transport protocol; an ESTELLE description of all the classes of the Transport protocol is being produced within ISO;
- description of the ISO Session service; contrary to the descriptions of the Network and Transport services which describe the end-to-end service behavior, this description expresses only the possible Session service primitive sequence at some particular Session Service Access Point;
- description of the ISO Session protocol; this description is complete as it supports all the fonctional units defined in the Session International Standard; the symetric synchronization mechanism, which currently has the draft addendum status, is not supported.

Among the other ESTELLE descriptions being currently produced or revised within SEDOS which did not yet reach the same level of maturity as the previous ones, one may underline:
- description of the ISO Presentation service; the overall architecture of the description as well as the specification of the connection establishment phase have been completed;
- description of the ISO Presentation protocol; the description is complete, but not yet parsed by the ESTELLE compiler;
- description of the FTAM protocol; the INITIATOR description has been completed and parsed, whereas the RESPONDER description is being currently written; this description follows the SPAG A111 profile recommendations and assumes a CASE/PRESENTATION lower layer service.

III.3. *Tools*

A considerable effort is presently being made in developing tools for ESTELLE. In fact, such tools are developed within two European projects (in the ESPRIT program), namely SEDOS and the SEDOS-ESTELLE-DEMONSTRATOR (July 1986 - June 1989). Prototype tools such as a syntax driven EDITOR, a COMPILER, a SIMULATOR, a VERIFIER and a DEBUGGER are the outcome of the SEDOS project; they are available either on SPS7 (a BULL microcomputer) or on SUN workstation; then, these tools serve as basis for an integrated and industrialized ESTELLE Work Station within the ESPRIT SEDOS-ESTELLE-DEMONSTRATOR project. Other but less important efforts in developing ESTELLE tools are also conducted in USA (National Bureau of Standards, Protocol Development Corporation), in Canada (University of Montreal, University of British Columbia) and in Japan.

The SEDOS-ESTELLE **EDITOR** is interactive and syntax driven. It is based on a meta-editor called MENTOR (produced at INRIA). The purpose of the editor is to provide help to build, modify, and beautify ESTELLE specifications. It takes as input (from a terminal or a file) an ESTELLE source text, parses the text and outputs error messages. If the text is syntactically correct, it builds an internal tree-form representation. The latter may then be unparsed to a nicely indented ESTELLE text. The basic MENTOR commands have been enhanced with a set of macro-commands tailored specially for ESTELLE.

The ESTELLE **COMPILER** is composed of two parts: the translator and a code generator. The translator takes as input the ESTELLE source text, detects syntax errors, and checks the static semantics. The translator output is an

Intermediate Form, composed of the Symbol Table, the Transition Table and a tree-form representation of transition blocks. The translator is generated using an automatic translator generator tool called SYNTAX (produced at INRIA). This Intermediate Form may then be taken as input by a code generator in order to produce code in C, ML or any other code for execution or interpretation. The C code thus generated may then be executed, once an operating system environment is created by a hand written C code.

An ESTELLE **SIMULATOR**, **ESTIM**, has been written in ML (a metalanguage developed by INRIA). ESTIM is an interpreter which allows to simulate ESTELLE specifications. It is based on an extended coloured Time Petri Net model. ESTIM takes as input the ML code generated by the compiler and interprets this ML code; this consists in interpreting the ESTELLE operations and the firing of transitions. The selection of transitions may be made by the user or randomly by ESTIM. ESTIM also provides an interactive access to display elements of the global state and modify them when adequate.

Specification of an ESTELLE **DEBUGGER** has been written. This specification is composed of three parts: the Debugger Scheme, the Command Interpreter, and the Error Processing. The Debugger Scheme describes the algorithms of the simulator in the debugger (based on the inductive operational definition of ESTELLE semantics) and the handling of Time parameters. The commands of the debugger help the user to navigate in the tree of module instances, allow the display of variables and the modification of those that are permissible; if desired, the user may also make choices to resolve non-determinism.

Additional effort (outside SEDOS) is being made by BULL. The outcome (to be available by the end of 1987) will cover: a new version of the compiler tool with useful extensions, e.g. the handling of "qualifying comments" and a preliminary version of the debugger tool.

IV. LOTOS

VI.1. *LOTOS definition*

The development of LOTOS (Language Of Temporal Ordering Specification) has been strongly guided by architectural principles, notably those that underly the OSI Reference Model and Standards. In particular the requirement that the FDT should allow that standards be specified in an implementation independent way has played a determinant role. This requirement has been fulfilled by four basic language design choices:
- the interaction concept,
- the temporal ordering principle,
- process abstraction, and
- the use of abstract data types.

The interaction concept is an advanced concept that replaces the traditional concepts of input and output. An interaction can be considered as a common activity of two or more processes in which information is established to which these processes can refer. Since the activity can be anything the interaction concept allows not only to model the traditional input and output of information (so called "value passing"), but also arbitrary forms of synchronization based on data values ("value checking"), and the non-deterministic generation of information ("value generation"). These forms of interaction, which syntactically and semantically are based on the same concept, have

proven to be all of extreme practicality in the development of specifications. Moreover they have provided insight in architecture accordingly.

The temporal ordering principle is used to define the externally observable behaviour of a process by the temporal ordering of its interactions with its environment. In so doing the semantics of the process definition by no way depends on the internal organization of the process itself, and so refrains from implementation orientedness. LOTOS specifies a process by a "behaviour expression" that defines an ordering of "events", an event being an atomic instance of interaction. LOTOS provides a set of temporal ordering "operators" that allow to combine and structure behaviour expressions in different ways and so can be used to express important architectural concepts. The algebraic nature of the language is brought out by the availability of set of transformation rules that allow to transform a behaviour expression in another with the same observational behaviour. The semantics of the language are defined similarly to that of Milner's CCS [23].

The concept of process abstraction roughly corresponds to the procedure concept in conventional programming languages. This concept allows to define a formal process with formal interaction points ("event gates") and formal parameters. Together with the possibility to use process instantiation recursively it provides a powerfull and indispensable specification structuring principle.

The use of abstract data types (ADTs) provides the complementary approach in achieving implementation independence. It allows to represent, for example, a set of connection endpoint identifiers in a completely abstract way rather than by way of a particular concrete data structure say, an array of integers. The LOTOS approach is based on an equational specification of data types with an initial algebra semantics and adopted much from ACT ONE [16]. The work on ADTs has also led to a collaboration with the CCITT Study Group X Working Party 3, and produced a common semantical kernel for LOTOS ADTs and the ADT part of SDL (CCITT's FDT) [10].

During its development several improvements have been applied to LOTOS. Suggestions predominantly came from the development of trial specifications in SEDOS (see next section). The development of the language itself, and thus the careful evaluation of proposed language improvements was however carried out in the framework of the study of verification methods and tools. We mention one language improvement that appeared to be of exceptional use, viz. the introduction of the possibility that more than two processes can engage in an event. This possibility opened the horizon for the introduction of the "constraint oriented specification style" (see next section), which allowed dramatic improvements in the quality, conciseness and ease of verification of specifications.

Another language improvement that is currently considered is the development of a graphical representation of LOTOS. This work again done in collaboration between ISO and CCITT involves many SEDOS experts.

As a final remark it should be noted that LOTOS is not restricted to the specification of OSI standards, but is developed as a general purpose specification technique and so is applicable to system specification in general. It should also be understood that LOTOS is capable of expressing behaviour at different levels of abstraction, and so can be used to describe implementations. Further indications on the future of LOTOS can be found in [9].

IV.2. LOTOS *Specification*

In the project the development of LOTOS specifications was mainly focussed on specifications of OSI Standards. This development has been guided by a number of principles:
- experience should be obtained with the description of a wide range of OSI standards,
- this wide range of standards should be described in a stylistically consistent way,
- the service and the protocol descriptions should be such that it is made as easy as possible to verify that the protocol provides the required service.

Experience with the description of a wide range of OSI standards was found necessary to check LOTOS and improve its definition (see IV.1.) on basis of proper evidence, but also to have realistic evidence about what size and complexity can be expected for specifications of these standards. In this line LOTOS has been applied to: HDLC, the IEEE LAN service, Connection-less Internetting Protocol, Network Service, Transport Protocol, Transport Service, Session Protocol, Session Service, Presentation Protocol and Transaction Processing Service. In addition descriptions were made of parts of FTAM, MTS of X.400, and a complete Flow Control by Latency Protocol.

This experience has shown that the complete description of a full size standard is a quite difficult and resource demanding task [24]. This is because the description on one hand has to deal with every aspect of the standard including all classes, functional units, options (these possibly classified as mandatory, conditional, etc) -all this in an implementation independent way!- and on the other hand has to preserve a clear and natural structure of the description. Given the fact that a specification can be structured in an infinite number of different ways, it can be understood that the many different functions of a standard, aggravated with the manifold and intertwined relationships between these functions, very much complicate the specification effort. One net result of this work was precisely that of disclosing an entirely new area of research, which can best be termed as that of "specification style".

For the above reasons it was decided to forward only the Formal Descriptions (FDs) located in the intermediate layers of the Reference Model, i.e. Network Service, Transport Protocol and Service, and Session Protocol and Service, to the appropriate ISO working groups, and to support these working groups to establish complete FDs in order to produce ISO Technical Reports. This work has also contributed to the quality of the informally described standards by reporting deficiencies that were detected on basis of the formal specification effort.

A consistent specification style over all layers of the OSI Reference Model was found necessary to avoid that a reader, e.g. an implementer, is unnecessarily burdened with different ways of expressing the same functionality. To achieve this goal several measures were taken. We mention two of them:
An interesting measure which appeared quite helpful was the development of an "interlayer standard" for the description of the Service Access Points and Service Primitives, preserving in particular a consistent definition style for all the abstract data types involved.

Another valuable measure was the adoption of the same specification style for all descriptions. Among the many different styles which have been exercised, such as "monolithic" or "resource oriented", it appeared that a "constraint

oriented" style very well suits the nature of OSI standards. This style allows to define the "abstract interface", i.e. the constraints on the ordering of Service Primitives at one Service Access Point, as a separate process. Since the abstract interfaces are identical for the service and the protocol, it allows the specifier to focus on the differences in these descriptions and to model these descriptions such that it can be more easily verified that the protocol provides the service and makes a correct use of the underlying service.

At this place it is worthwile note that LOTOS has also been applied to other fields of technology, in particular to the development of Computer Integrated Manufacturing architectures [4].

IV.4. *LOTOS tools*

Since LOTOS is a quite new and avanced technique, its tool developments could not build on experience available from elsewhere. Therefore the following approach was adopted:
- any tool development should be preceded by the development of a sound formal theory that must underly the tool,
- the feasibility of the tool should be investigated before attention is given to the efficiency of the tool,
- all tool developments should be placed in the framework of a consistent "toolkit" architecture.

Given the fact that the LOTOS tool developments started from scratch, these conditions certainly could not be met for all projected tool developments. It can be observed, however, that the project provided more and better results than was originally expected.

In line with the above approach, and given the fact that LOTOS is in first instance a formal description language, the first generation of tools to be tackled should support the specification phase of the design process. In this context some experimental tools were developed early in the project:
MELO (Mentor LOTOS Editor): a structure editor with a built-in LOTOS syntax. It is built on the Mentor system originally produced by INRIA and now distributed by SEMA.
BELASI (Behavioural Language Simulator): a simulation tool written in C-Prolog. It implements the dynamic semantics of LOTOS and allows simulation of a specification by single-stepping through a LOTOS behaviour tree. The ADT aspect is rudimentarily supported.
STAR+ : a simulation tool similar to BELASI, however less developed. It is derived from STAR, a simulation tool for CSP, and implemented in Lispkit.

Further developments of this first generation of tools resulted in an integrated toolset [22] which is written entirely in the 'C' programming language and is hosted on BSD 4.2 UNIX (Figure 3).

EDITOR: the output of this tool is an ASCII textfile containing standard LOTOS text. The editing phase currently relies on any general purpose Unix editor for registration and modification of a specification. The MELO prototype is not thought to be suitable for other than research.

FRONT-END: The Front-End is a collection of tools which perform the following functions:
- validation of correct syntax conforming to ISO DP8807 section 6,
- validation of correct static semantics conforming to ISO DP8807 section 7.2,

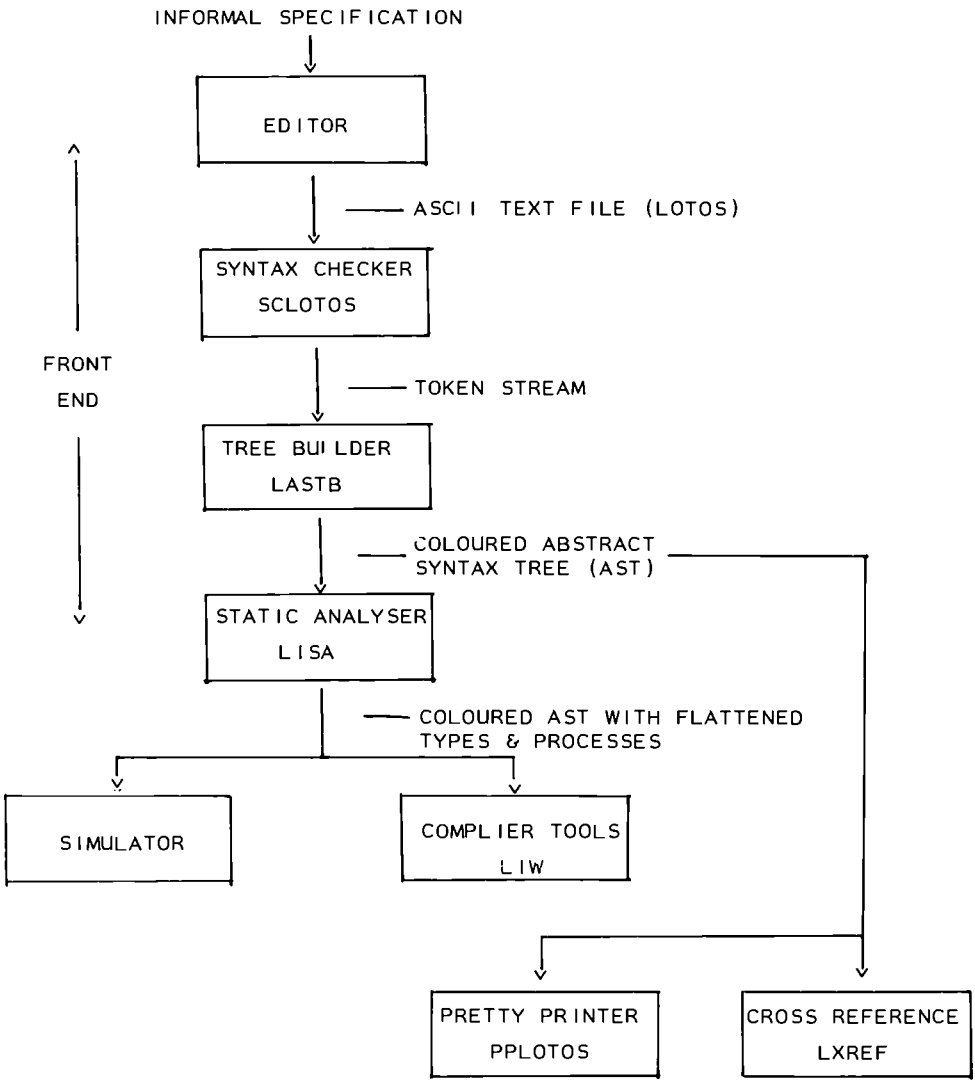

INFORMAL SPECIFICATION
EDITOR
ASCII TEXT FILE (LOTOS)
SYNTAX CHECKER
SCLOTOS
FRONT
END
TOKEN STREAM
TREE BUILDER
LASTB
COLOURED ABSTRACT
SYNTAX TREE (AST)
STATIC ANALYSER
LISA
COLOURED AST WITH FLATTENED
TYPES & PROCESSES
SIMULATOR
COMPLIER TOOLS
LIW
PRETTY PRINTER
PPLOTOS
CROSS REFERENCE
LXREF

FIGURE 3

- insertion of Abstract Data Type (ADT) definitions from a standard ADT library,
- flattening of LOTOS ADT's as described in ISO DP8807 section 7.3.4.1.,
- construction of an intermediate representation of the specification in the form of a coloured Abstract Syntax Tree.

The input to the Front-End is the standard LOTOS text produced by the Editor. The output conforms to a standard interface for the remaining tools in the toolset. At this interface the basic properties of the specification have been checked and the specifications has undergone a transformation so that it contains only a single flattened LOTOS data type. The components in the Front-End are SCLOTOS, LASTB and LISA:
SCLOTOS (Syntax Checker LOTOS) is built on the Yacc system distributed with UNIX. The accepted syntax covers the full specification of ISO DP8807, section 6, plus minor extensions that can be switched off.
LASTB (LOTOS Abstract Syntax Tree Builder) accepts a syntactically correct LOTOS text and outputs a coloured abstract syntax tree used as a common representation for the rest of the toolset. LASTB is based on EL PRADO, a methodology and library to handle attributed abstract syntax trees.
LISA (LOTOS Integrated Static Analyser) performs the combined functions of static semantics checking, ADT library insertion, and ADT flattening.

SIMULATOR: The functionality of the simulator is a further extension of the BELASI tool [17] and performs the following functions:
- generation of an ADT Term Rewrite System,
- evaluation of an ADT Term Rewrite System,
- communication tree walker,
- menu computation and selection,
- term analysis and display,
- state information display.

This tool (and its front ends) is in a quite advanced stage and will be demonstrated during the ESPRIT Technical Week.

COMPILER: The compiler is a collection of tools which are, by preference, referred to as the LOTOS Implementation Workbench, or LIW. The collection currently consists of:
- an interactive computer aided source to source transformer which performs the translation of a high level specification to a more machine oriented done,
- a simplifier that performs source to source transformation to simplify the input to the actual compiler,
- an abstract data type compiler that generates a rewrite system to produce normal forms and an equality test,
- a temporal ordering compiler that handles processes that are translated into co-routine like pieces of C-code that interact with a kernel, or run time support system, to implement the multiway synchronization semantics of LOTOS.

Although important progress has been made, allowing to thoroughly assess the approach taken, the compiler development is still in an early stage of development.

PPLOTOS: PPLOTOS (Pretty Printer for LOTOS) is a schema driven pretty printing system for the display of Abstract Syntax trees compatible with EL PRADO or LASTB. The output is LOTOS text which may be displayed or filed.
LXREF: LXREF (LOTOS Cross Reference Generator) is an early prototype tool that correlates identifier occurrences over a LOTOS text.

Other implementation support tools
The above described toolset already includes elements of tools that support the implementation phase of the design process, notably the compiler.

Compiler, tester and verifier are considered as a second generation of tools which predominantly support the implementation phase. The development of these tools requires the development of more theory and models to clearly identify the formal relationship of an implementation to its specification. Early results are available in [7].

VERIFIER
Early work in this area concentrated on the study of verification models, algorithms and mechanisms for LOTOS and the specification of prototype verification tools. Reports are available that summarize the possibilities and limitations of several verification approaches. Currently a set of prototype verifiers for parts of LOTOS are implemented or available. It is expected that all prototypes are available by the end of the project, including:
- a verifier for observational equivalence written in Quintus Prolog on a Sun workstation: this work is completed.
- a PROLOG prototype for checking the persistency of ADT specifications: this work is in progress.
- the development of a tool (TILT) that translates LOTOS in a labelled transition system which may subsequently be input to CESAR and ECRIN tools: this work is in progress.

TESTER
Also for the development of test tools the effort first concentrated on the development of a sound theoretical basis for testing. A set of theoretical results are now available [8] and early work on prototype specification and implementation is in progress, in particular the derivation of test sequences from specifications using the functional language Twentel which is available under VAX Unix and MS-DOS.

V. CONCLUSION

This paper has given some of the results of the ESPRIT-SEDOS project, which include:
- the development of the ISO FDTs ESTELLE and LOTOS,
- their application to the specification of real and complex ISO protocols,
- the development of prototype tools for supporting these FDT based design of distributed systems.

The readers must refer to the bibliography [25] for more explanations and examples.

ACKNOWLEDGEMENTS
The success of the SEDOS project is due to the combined effort of many experts that actively contributed to the project.
The authors would like to express their sincere thanks to all participants involved in the SEDOS project: first to the task managers, for their high quality leadership and their day to day responsability for the progress of the work, J.P. Ansart (B), J.P. Courtiat (B1), P. Azema (B2), V. Chari (B3), A. Tocher (C1), J. Quemada (C1), E. Brinksma (C2), A. Marshall (C3), R. Neely (C3); and second to all the SEDOS participants of the LAAS-CNRS, the University of TWENTE, ADI, ICL, BULL, the University of CATANIA, the Polytechnics of MILANO and MADRID, HMI and the Technical University of BERLIN.

BIBLIOGRAPHY

[1] J.P. Ansart, V. Chari, D. Simon, "From formal description to automated implementation using PDIL", Protocol Specification, Testing and Verification, III, North-Holland, 1983, H. Rudin - C. West Editors.

[2] J.P. Ansart et al, "Software tools for ESTELLE", Sixth Int. Workshop on Protocol Specification, Testing and Verification, Montreal, June 10-13,1986.

[3] J.M. Ayache, J.P. Courtiat, "LC/1, A specification and implementation language for protocols", Protocol Specification, Testing and Verification, III, North-Holland, 1983, H. Rudin - C., West Editors.

[4] F. Biemans, P. Blonk, "On the formal specification and verification of CIM architectures using LOTOS", Computers in Industry 7, 1986, pp.491-504.

[5] T.P. Blumer, D.P. Sidhu, "Mechanical verification and automatic implementation of communication protocols", IEEE T on Software Eng., vol.SE-12, n°8, August 1986.

[6] G.V. Bochmann "Finite state description of communication protocols", Conf. Computer Network Protocols, Liège, 1978, also in Computer Networks 2, 1978, pp.361-372.

[7] E. Brinksma, G. Scollo, "Formal notions of implementation and conformance in LOTOS", Twente Univ., Memorandum INF-86-13, Dec. 1986.

[8] E. Brinksma, G. Scollo, C. Steenbergen, "LOTOS specifications, their implementations and their tests", Proc. 6th IFIP WG6.1 Workshop on Protocol Specification, Testing and Verification, Montreal, June 1986, North Holland, pp.349-360.

[9] E. Brinksma, G. Scollo, C.A. Vissers, "Experience with and future of LOTOS as a specification language", Proceedings of the Third CCITT SDL Forum, the Hague, Netherlands, 6-10 April 1987, North-Holland.

[10] CCITT/SGX/3-1, "Recommendation Z104, Functional specification and description language", June 1986.

[11] J.P. Courtiat, P. Dembinski, R. Groz, C. Jard, "ESTELLE : un langage pour les algorithmes distribués et les protocoles", Techniques et Sciences Informatiques, vol.6, n°2, 1987.

[12] P. Dembinski, S. Budkowski, "Defining delays in a time-independent model for ESTELLE", Report FDT 7581, Agence de l'Informatique, 1985.

[13] M. Diaz, "Modeling and analysis of communication and cooperation protocols using Petri net based models", Tutorial paper, Computer Networks, vol.6, n°6, December 1982.

[14] M. Diaz, Ch. Vissers, J.P. Ansart, "SEDOS, Software environment for the design of open distributed systems", ESPRIT Week, Brussels, September 1985.

[15] M. Diaz, J.P. Courtiat, P. Dembinski, E. Brinksma, "Formal description techniques in SEDOS", ESPRIT 1986, North Holland, DGXIII Editors, 1987.

[16] H. Ehrig, B. Mahr, "Fundamentals of algebraic specification", Springer Verlag, Berlin 1985.

[17] P. Van Eijk, "A comparison of behavioural language simulators", Proc. 6th IFIP WG6.1 Workshop on Protocol Specification, Testing and Verification, Montreal, June 1986, North-Holland, pp.85-96.

[18] H.J. Genrich, K. Lautenbach, "The analysis of distributed systems by means of predicate/transition nets", Lect. Notes in Computer Sciences, vol.70, Springer Verlag, 1979, pp.123-146.

[19] D.I. Good, R.M. Cohen, " Verifiable communications processing in GYPSY", Proc. of COMPCON 78, IEEE, September 1978.

[20] ISO-DIS9074, ISO/TC97/SC21/WG1-FDT/SG-B, "ESTELLE, a formal description technique based on an extended state transition model", December 1986.

[21] ISO-DIS8807, ISO/TC97/SC21/WG1-FDT/SC-C, "LOTOS, a formal description technique based on the temporal ordering of observational behaviour", December 1986.

[22] A. Marshall, "A prototype integrated toolset for LOTOS", C3 Progress Report, ESPRIT/SEDOS/C3/15, STC Tech. Ltd., Newcastle-under-Lyne, England, April 1987.

[23] R. Milner, "CCS, a calculus of communicating systems", LNCS 92, Springer-Verlag, 1980.

[24] G. Scollo, C.A. Vissers, A. di Stefano, "LOTOS in practice", Proc. IFIP 86, 10th World Congress, Dublin, Sept. 1986, pp.869-875.

[25] SEDOS Project, "SEDOS publications and reports", November 1984 to October 1987.

[26] F.D. Smith, C.H. West, "Technologies for network architecture and implementation", IBM J. of Res. and Develop., vol.27, n°1, January 1983.

[27] C.A. Vissers, "Standardization of formal description techniques for communication protocols", in H.J. Kugler (ed), Proc. IFIP Congress 1986, Dublin, September 1986, pp.329-333, North-Holland, 1986.

[28] Ch. Vissers, L. Logrippo, "The importance of the concept of service", 5th Int. Workshop on Protocol Specification, Testing and Verification, Toulouse, North-Holland, M. Diaz Editor, 1986.

Project No. 881

A NEW LANGUAGE TO DESCRIBE ANALOG CIRCUITS *

C. van Reeuwijk and M.G. Middelhoek.

Delft University of Technology.

ABSTRACT

At the moment there are some circuit description languages in use (e.g. the SPICE input language), but they have been designed with a special purpose in mind and therefore lack scope. In the recently developed functional programming languages computational problems are described by the relations that must hold. The functional description method is preeminently suitable for the description of circuits; it allows the creation of a very clear circuit description language that can be manipulated by computer. As an illustration we will discuss two experimental circuit description languages. We will also describe the possible applications of such a language.

1. Introduction.

At the moment there are large areas in analog circuit design where little formal theory is available. In these areas design must be based on informal knowledge like experience of designers and rules of the thumb. For a better understanding of analog circuit design it is necessary to develop more formal theory, as is done at the moment at Delft University of Technology. To help the promotion of te new theory it is tried to incorporate the knowledge in computer algorithms. The final goal of this development is a silicon compiler for analog circuits.

An important requirement for further development of the theory is a good notation to describe circuits. As will be shown in this paper, existing notations like schematic drawings are inadequate, and need to be replaced by more advanced notations.

To a large extent the comments on analog circuit design also apply to digital circuit design, although in this area theory is further developed. The notation described in this paper is also suitable for the description of digital circuits.

* This research was done for Esprit project 881 (FORFUN), and is partly funded by the European Community.

2. Existing notations for circuit description.

The most frequently used notation is the schematic drawing, for example Figure 1.

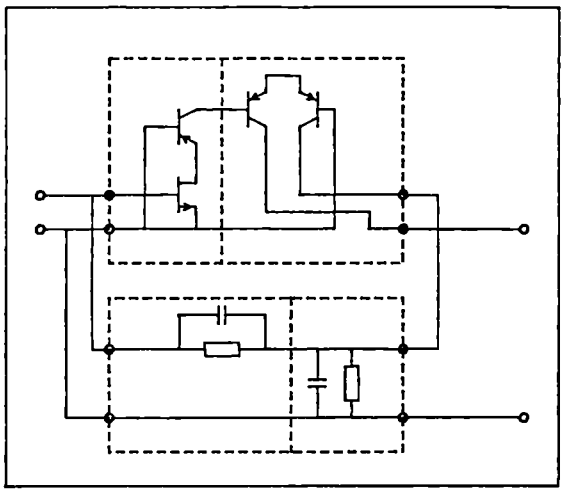


Figure 1.

An example of a schematic drawing: the signal path of a current amplifier.

Apart from its obvious advantages (well known, good standardization) there are some important disadvantages: first, much important information is left out: the hierarchy of the circuit, the function of the various blocks, the interconnection pattern that is used. Also, important remarks about the details of the circuit are difficult to include in the description. All this information must be inferred by the reader, requiring much knowledge of electronic circuits, or must be given in a companion description text, which is awkward.

Next to schematic drawings there are a few textual notations in use to describe circuits. Unfortunately these languages are only intended for a special purpose, and are therefore unsuitable for general circuit description. A typical example of such a language is the input language of SPICE. An other common disadvantage of current circuit description languages is that they are inspired by imperative languages like Pascal or C. This is unfortunate since these circuit description languages end up describing a way to build the circuit, and not its static structure. Finally, most of these languages often have an inconvenient and vaguely defined syntax. Again SPICE is an example.

3. New languages.

A recent development in programming languages (that is, languages that describe computations) is the emergence of declarative languages. Contrary to imperative languages they do not describe computations as a list of commands that must be executed in a given order, but as a set of relations that must hold. It is left to the language implementation to determine the way the problem must be solved. At present, theory is not sufficiently developed to allow the description of arbitrary relations in these descriptions. Therefore, all current languages impose restrictions on the class of relations that are allowed. For example, in Prolog the relations are restricted to the predicates of predicate calculus (see [Cl81]).

An important class of declarative languages is that of functional languages, where the relations must be described as functions. This restriction ensures that a more robust implementation is possible. Examples of functional languages are Hope and Miranda. These functional languages are based on the theoretical concept of lambda-calculus (see e.g.[Ba81]). For logic programming languages a slight generalization of lambda-calculus is necessary. In [Bo86] a proposal for such a generalization is given, it is called "beta-calculus".

The concept of declarative languages is very suitable for circuit descriptions since it allows one to describe exactly what is necessary - the relations between the components. Both the notations based on lambda-calculus and beta-calculus are useful; lambda-calculus for circuits where one wants to suggest direction of information flow (e.g. an amplifier), beta-calculus for circuits where one does not want do that (e.g an impedance network). Development of suitable languages and language implementations is still underway, but as an illustration it is useful to discuss two prototype languages, FUN and SYMNET.

4. FUN.

The development of FUN (FUNctional circuit description language) was inspired by the proposals of Boute [Bo84] and the article of Backus [Ba78] on FP. FUN uses the language constructs of FP to describe analog electronic circuits, their devices and interconnections. The devices are represented by objects, while the interconnections are represented by functions that map devices to devices. There is a standard set of interconnection functions available, and it is possible to define new interconnection functions. This can be done by combining existing functions (which is simple), or by defining an entirely new interconnection function (which is difficult but possible). The following circuit description can serve as an example of this method. It describes the structure of the current amplifier of Figure 1:

```
# A current amplifier, consisting of two twoports an active
# part and a feedback loop, connected by the
# interconnection function "piso". The interconnection
# function "piso" connects two twoports with inputs
# parallel, and outputs in series.
def current_amplifier =
   piso : <active_part, feedback_loop>

# The active part consists of thee stages.
def active_part =
   three_stage : <J1, T1, <T1, T1>>

# Cascade of a common-source stage, a common-base stage,
# and a differential-pair stage with its output terminals
# inverted.
def three_stage =
   /casc~[cs~1, cb~2, outinv~diffst~3]

# A current divider with its inputport and outputport
# interchanged.
def feedback_loop =
   rev~idiv : <par:<R1, C1>, par:<R2, C2>>
```

This example clearly illustrates the advantages of FUN, when compared to conventional circuit description languages:

- There is a close match between such language constructs as the interconnection functions parallel (par) and series (ser) and the structural concepts to be expressed. It resembles the way designers talk about circuits.

- The readability of the functional circuit description can surpass that of schematics, because of its better ability to express the hierarchy of a circuit.

- The language has very interesting formal properties, which are based on its reduction semantics: an expression can be successively reduced to simpler expressions to yield a final "normal form expression". This means that reasoning about a circuit description can proceed by expression transformation.

These properties show that a functional language is suitable as a framework for a circuit description language. However, FUN also has its drawbacks. The main problem is that in FUN it is difficult to define new interconnection functions and that it is impossible to define a standard set that is suitable for a majority of cases. This problem is even more important for non-hierarchical circuits, since each of them would require a special interconnection function.

5. SYMNET.

Based on the experiences with FUN and new theoretical results from Boute [Bo86] an other prototype language was developed: SYMNET (for SYMbolic Network language). In SYMNET the idea to base the language on FP is abandoned, and more traditional functional languages are taken as example. The main advantage of SYMNET is that in this language it is much easier to define arbitrary new interconnection functions.

The simplest way to describe the language is to give an example. The definitions below describe the most abstract levels of the current amplifier of Figure 1.

```
// A current amplifier, consisting of an active part and a
// reversed feedback loop, connected with inputs parallel,
// and outputs in series.
def current_amplifier in =
   piso active (rev feedback) in;

// the active part is a nullor
def active in =
   nullor in;

// The feedback loop is a two-port with the given Laplace
// transform.
def feedback in =
   laplace_twoport "p/((p-1)*(p-1))" in;

// The implementation of the piso combinator. It defines a
// two-port consisting of two twoports given as parameters
// that are connected with their inputs in parallel, and
// their outputs in series.
def piso A B < in, <out+, out- > >
{
   A < in, < out+, i > >;
   B < in, < i, out- > >;
}

// A new name for the type of a twoport
typedef twoport = < < pin, pin >, < pin, pin > >;

// the piso is given two twoports as parameters and defines
// a twoport
type piso : twoport->twoport->twoport;
```

The comments in this description should explain the purpose of the various definitions. Note the wealth of information that can be given:

- Hierarchy.
- The interconnection patterns used.
- Structure of the component terminals.
- Direction of data flow, and the fact that there is a data flow.

6. Applications.

The circuit descriptions of FUN and Symnet are general enough to allow many interpretations. It is simple to convert them to input cards for SPICE, but it is also possible to generate net-lists (lists of components and interconnection wires) from which the layout of the circuit can be generated.

Regarding the many possible interpretations Boute [Bo84] has proposed to describe these interpretations as functions in a functional language. Such a semantic function or meaning function takes a circuit description as input and returns a description in a different domain of interpretation as result.

When the domain of interpretation is also the domain of circuit descriptions the semantic function is called a transformation function. Such transformation functions are useful to describe synthesis routines for automated circuit design.

7. Prospects.

The ideas described in this paper are currently subject of ESPRIT Project FORFUN. The final goal of this project is a design environment wherein both analog and digital circuits are described with the declarative notation outlined in this paper. The design environment will support circuit analysis, layout design etc. with appropriate meaning functions. The environment will allow easy definition of new meaning functions by providing a suitable functional language to define them in. The synthesis algorithms for analog circuits that are currently developed at Delft University of Technology (e.g. the noise optimization algorithms of Stoffels [St86]) will be incorporated in the environment as transformation functions.

8. Conclusion.

Although we do not have the illusion that these languages will replace schematic drawings as the main notation for circuit descriptions, we are confident that there is a wide area of applications. Current research suggests that these languages are suitable for human design of structured circuit and for automated design. The same concepts may be applied to other areas, ranging from layout design to the design of mechanical systems.

References

Ba78. J. Backus, "Can Programming be liberated from the von Neumann style? - A functional style and its algebra of programs," Comm. of the ACM Vol. 21 pp. 613-641 (Aug. 1978).

Ba81. H. Barendregt, The lambda calculus, its syntax and semantics, North Holland, Amsterdam (1981).

Bo84. R.T. Boute, System Semantics Applied to Digital and Analog Circuits, University of Nijmegen, Department of Computer Science, Nijmegen (1984).

Bo86. R.T. Boute, Syntactic constructs for the Description of bidirectional systems, University of Nijmegen, Department of Computer Science, Nijmegen (1986).

Cl81. W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer-Verlag, Berlin (1981).

St86. H. Stoffels and E.H. Nordholt, "Automated noise optimization in amplifiers," Proc. of symp. on Circuits and systems, pp. 1139-1141 IEEE, (May 5 1986).

Project No. 1033

A COMPOSITIONAL METHOD FOR THE DESIGN AND PROOF OF ASYNCHRONOUS PROCESSES

R. J. Cunningham†, A. Nonnengart††, A. Szalas†

Asynchronous processes arise naturally in real time environments where there is human-computer interaction, but formalisms for describing asynchronous processes and the ways of reasoning about them are not well understood. Abstract programming systems which assume asynchrony display a variety of of message-passing and stream architectures in which a process receives input through a mail-box or set of channels. These systems can be contrasted with programming systems like CSP and Ada which use the primitive notion of a rendevous to provide a synchronous basis for communication. In this paper we develop a compositional proof theory for the safety properties of asynchronous systems.

† Department of Computing, Imperial College of Science and Technology
†† FB Informatik, University of Kaiserslautern.

1. INTRODUCTION

Asynchronous processes arise naturally in real time environments where there is human-computer interaction, but formalisms for describing asynchronous processes and the ways of reasoning about them are not well understood. Programming systems which assume asynchrony display a variety of of message-passing and stream architectures in which a process receives input through a mail-box or set of channels. These systems can be contrasted with programming systems like CSP [6] and Ada, which use the primitive notion of a rendezvous [8] to provide a synchronous basis for communication. In this paper we develop a compositional proof theory for the safety properties of asynchronous systems. The advantage of the compositional style is that there is a potential for application in a design procedure which produces correct parallel programs from correctly designed parts.

Our work arises in the context of an Esprit project to provide a formal basis for asychronous computer system architectures and uses the architecture of the prime contractor as motivation. This was developed by Hemdal [4] and takes local asynchrony to the extreme. There are no shared variables between processes, processes may exist indefinitely, and the abstract style of programming which is encouraged is to introduce as many processes as is warranted for conceptual partitioning of the task. A system is usually constructed as a fixed collection of processes linked by certain directed communication channels. Our interest in this paper is the development of a suitably formal basis for constructing such a system. We suggest a proof system for the composition of asychronous processes which is compatible with Hemdal's style, although not restricted to it. Indeed, it leads us towards a more uniform generalisation of such architectures than is at present available.

The underlying intuition of an elementary process which we use here is that of a non-deterministic sequential state machine with a finite set of *control states* but potentially infinite *data states*. The input and output for this machine are communication channels to other processes. A conventional sequential program is a deterministic example of a single process, one in which each control state corresponds to a node of its flow chart and the subsequent state-changing action is selected deterministically. Dijkstra's guarded commands [3] can be used to describe a well-structured non-deterministic sequential process. We do not restrict ourselves at the outset to well-structured sequential processes because different styles are possible and concrete realisations of abstract asynchronous processes trade performance for

simplicity of structure. Additional complexity of the reasoning should be reflected in the rules of composition.

The traditional formal treatment of programs by intermediate assertions in an applied first order logic can be translated into assertions about the data at control states in the state-machine notion of a process. In order to consider the construction of processes we treat suitably labelled fragments of a potential control-flow graph as elementary processes, annotate them to identify the essential elements, and express different forms of combination by formal operators. An annotated graph thus becomes a *formal process* of our logical system. The usual *formulae* of multi-sorted first-order predicate logic are assumed. In effect, a *formal process* becomes a special sort in an applied logic of processes.

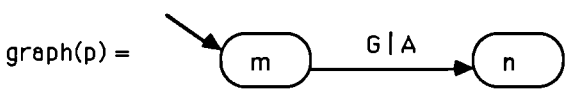
For the purposes of this paper we introduce a series of operators for process composition with a fairly primitive graphical style. Then we give a collection of proof rules which are applicable to a reasonably large class of processes. Some related work is mentioned briefly later.

2. A SYNTAX OF PROCESSES

Below we define the syntax of processes, together with some notions useful in the next sections of our paper. In particular we define:

- graph(p) denoting a graph representing p,
- var(p) denoting the set of variables of process p,
- nodes(p) denoting the set of nodes of p,
- init(p)$\in$ nodes(p) denoting the initial node of p,
- term(p)$\subset$nodes(p) denoting the set of terminal nodes of p,
- $\sim_p$ denoting a binary relation on nodes(p).

2.1. Let G be an open formula (sometimes called guard) and A an atomic action (skip, substitution, send t to q, receive x). Then



is a graph of process p consisting of the transition leading from m to n and executing G|A. In such a case:

- var(p) is the set of all variables appearing in A and G,
- nodes(p) = {m,n},
- init(p) = m,
- term(p) = {n},.
- $\sim_p$={(n,n), (m,m)}

2.2. If p and r are processes such that nodes(p)∩nodes(r)=Ø then p+r and p;r are processes such that:

- graph(p+r) = graph(p)∪graph(r) = graph(p;r),
- var(p+r) = var(p)∪var(r) = var(p;r),
- nodes(p+r) = nodes(p)∪nodes(r) = nodes(p;r),
- init(p+r) = init(p) = init(p;r),
- term(p+r)=term(p)∪term(r); term(p;r)=term(r),
- $\sim_{p+r} = \sim_p \cup \sim_r \cup \{(init(p),init(r))\}$
- $\sim_{p;r} = \sim_p \cup \sim_r \cup \{(k,init(r)) \mid k \in term(p)\}$

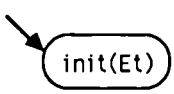
2.3. if p,r,...,s are process identifiers then the sequence

$$p = E_p, r = E_r, \ldots\ldots, s = E_s$$

of equations defines processes p,r,...,s, where E_t (t=p,r,...,s) is an expression obtained by application of + and ; to atomic processes and/or processes p,r,...,t such that t does not appear as the leftmost component of E_t (e.g. the leftmost component of r;p is r). In such a case for t=p,r,...,s,

- graph(t) = graph(E_t),
- var(t) = var(E_t),
- nodes(t) = nodes(E_t),
- init(t) = init(E_t),
- term(t) = term(E_t) - {init(t)},
- $\sim_t = \sim_{E_t}$

where by graph(E_t) we mean the graph defined by expression E_t in which t is replaced by a single node of the following form:



2.4. If p,r,...,s are processes such that for all $t_1, t_2 \in \{p,r,...,s\}$, $t_1 \neq t_2$, $nodes(t_1) \cap nodes(t_2) = \emptyset$ and $var(t_1) \cap var(t_2) = \emptyset$, then p||r||...||s is a program in which:

- graph(p||r||...||s)=graph(p)∪graph(r)...∪graph(s),
- var(p||r||...||s)=var(p)∪var(r)∪...∪var(s),
- nodes(p||r||...||s)=nodes(p)∪nodes(r)∪...∪nodes(s),
- init(p||r||...||s) ∈ {init(p), init(r),..., init(s)},
- term(p||r||...||s)=term(p)∪term(r)∪..∪term(s),
- $\sim_{p||r||...||s} = \sim_p \cup \sim_r \cup ... \cup \sim_s$.

We assume a priority order +;|| on operators with + binding most and || binding least, so that p+q+r || s+t || u means ((p+q)+r)||(r+s+t)||u.

Note that some context conditions are necessary, e.g. if a process does not appear in a context containing || then neither *send* nor *receive* can appear in actions, etc.

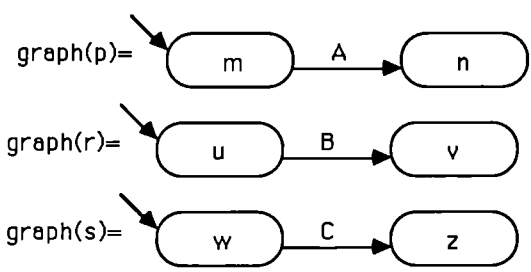
Let p be a process (program). By $\approx_p$ we shall mean the least equivalence relation on nodes(p) containing $\sim_p$. An intuitive meaning of $n \approx_p m$ is "n and m denote the same node of p".

By a *transition graph* of p we mean the graph obtained from graph(p) by identifying all equivalent nodes (w.r.t. $\approx_p$).

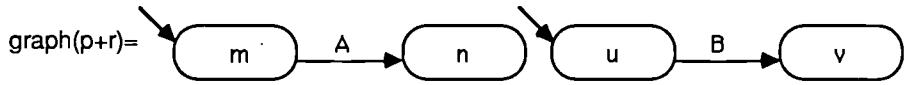
The following example shows an informal meaning of these notions.

Example:

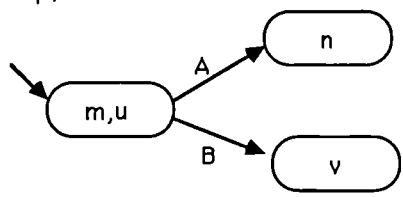
If processes p, r and s are such that:

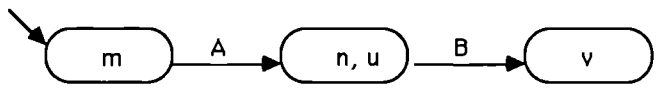


then graph(p+r) (as well as graph(p;r)) takes the form:



transition graphs of p+r and p;r take the forms:

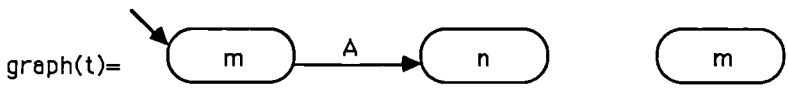




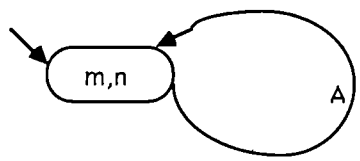
If t is defined by the equation

$$t = p;t$$

then



and, since $n \approx_t m$, the transition graph of t takes the form:



Note that an instance of + (;) may serve the purpose of non-deterministic choice (sequential composition) in the Dijkstra guarded command style. The operator || is intended for the parallel composition of processes. The possibility of defining processes by equations allows to define loops in graphs. It also serves the purpose of modularization.

3. AN OPERATIONAL SEMANTICS

For the purpose of exposition let us introduce the data type QUEUES of natural numbers. This data type is two-sorted, the defined sort Qu and

the natural numbers ω. We assume the following operations:

empty: $\rightarrow$ Qu,
insert: $\omega \times$ Qu $\rightarrow$ Qu,
delete: Qu-{empty} $\rightarrow$ Qu,
front: Qu-{empty} $\rightarrow$ ω.

The model of data type QUEUES assumed in our paper is the standard one. Thus the operations satisfy the following properties that can be used in reasoning about programs:

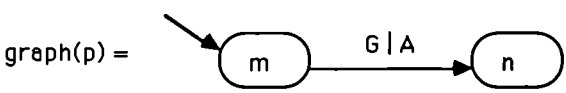
Q1. $\forall e \forall q(q \neq empty \rightarrow delete(insert(e,q)) = insert(e,delete(q)))$,
Q2. $\forall e(e = front(insert(e,empty)))$,
Q3. $\forall e \forall q(q \neq empty \rightarrow front(insert(e,q)) = front(q))$,
Q4. $\forall e \forall q(insert(e,q) \neq empty)$,
Q5. $\forall e(empty = delete(insert(e,empty)))$.

The data type QUEUES is introduced in order to fix a semantics for the inter-process communication mechanism. Nevertheless, the proof system we give in this paper is independent of a concrete data type. In order to work with multisets or stacks, one has only to give new axioms instead of Q1-Q5. The situation is analogous to that in classical first-order logic, where one can have different theories by considering different sets of specific axioms.

For the sake of simplicity we assume that the only individual data elements which can appear in processes are natural numbers and that the messages are simply natural numbers. In order to deal with different types of messages, one can introduce other sorts into data type QUEUES. We further asssume that with each process there is associated a "system" variable Q representing a queue of messages. By a *set of computations of a process* we understand the set of (possibly infinite) sequences of valuations labelled by nodes. The set is defined inductively as follows.

3.1. Sequential Computations

1. Let p be a process represented by the following transition graph:



and let v be a valuation of local variables of p, i.e. v: var(p)$\rightarrow\omega$. As usual, v can be extended to terms and formulae. The computation of p with v as an initial valuation of variables and Q as initial value of the message queue is defined as follows:

(i) if v(G)=*false* then the only computation of p is the one-element sequence comprised of the tuple <m,v,Q>;
(ii) if v(G)=*true* and A is the skip instruction then the only computation of p is the sequence (<m,v,Q>,<n,v,Q>),
(iii) if v(G)=*true* and A is a substitution of the form $x_1,...,x_k:=t_1,...,t_k$ then the only computation of p is the sequence (<m,v,Q>,<n,v',Q>), where v' differs from v at most on the variables $x_1,...,x_k$, and $v'(x_1)=v(t_1),..,\ v'(x_k)=v(t_k)$.

Note that because of the syntactic restrictions A cannot be of the form either *receive x from r* or *send* t *to r*

2. Assume p is defined by an expression. Then the set of computations of p is the least set of sequences satisfying the following conditions:

(i) the first element of any sequence takes the form <k,v,Q> where k=init(p);
(ii) if an element in a sequence is of the form <n,v',Q> then its successor, if it exists, takes the form <m,v",Q> provided that the transition graph of p contains a transition from n to m such that its Boolean guard is true and v" is obtained from v' according to case 1 above.

3.2. Parallel Computations

If P is a program of the form p||r||...||s, then any computation of P consists of elements of the form:

$$<m_p m_r...m_s, v_p v_r...v_s, Q_p Q_r...Q_s>$$

where $m_p, m_r,...,m_s$ describe the control flow of processes p,r,...,s, $v_p, v_r,...,v_s$ are valuations of local variables of p,r,...,s, and $Q_p, Q_r,...,Q_s$ are queues of messages waiting for receipt. We adopt the usual model of interleaving actions. Thus the set of computations of P is the least set containing sequences satisfying the following conditions:

(i) the first element of any sequence takes the form

$$<n_p n_r...n_s, v_p v_r...v_s, Q_p Q_r...Q_s>,$$

where n_p=init(p), n_r=init(r), ..., n_s=init(s), $v_p,v_r,...,v_s$ are initial valuations of variables of p,r,...,s and $Q_p,Q_r,...,Q_s$ represent the initial contents of respective queues,

(ii) if the element $<m_p...m_t...m_s,v_p...v_t...v_s,Q_p...Q_u...Q_t...Q_s>$ appears in a sequence, then its successor (if it exists) is of the form:

$$<m_p...m_t'...m_s,v_p...v_t'...v_s,Q_p...Q_u'...Q_t'...Q_k>$$

provided that the transition graph of t contains a transition with true guard, leading from m_t to m_t' such that:

- if the action performed during the transition is a substitution or skip, then v_t' is obtained as in 1(ii) with $Q_u'=Q_u$, $Q_t'=Q_t$;
- if the action performed during the transition is of the form *send e to u* then $v_t'=v_t$, $Q_t'=Q_t$, and Q_u'=insert(v_t(e),Q_u);
- if the action performed during the transition is of the form *receive x*, and $Q_t\neq$empty then v_t' differs from v_t at most on x, and $v_t'(x)$=front(Q_t), $Q_u'=Q_u$, and Q_t'=delete(Q_t).

A transition in which action is of the form *receive x* cannot be chosen when the respective queue is empty.

Note that in the above we do not consider delays of messages. The proof system we give, however, is sound even if one allows such delays.

4. THE PROOF SYSTEM

4.1. Syntax of the Logic

1. The usual syntax rules for classical first-order formulae augmented with additional syntax rule that if *n* is a node of a process, then *at* n is a formula,

2. If p is a program, A is a classical first-order formula and B is a formula of the form defined in 1 then {A}p{B} is a formula.

An intuitive meaning of {A}p{B} is the following:

"if p initially satisfies A then p always satisfies B"

4.2. Semantics of the Logic

We define the satisfaction relation $\models$ as follows. Let P=p||r||...||s be a

program, let $n_p, n_r, \ldots, n_s$ be nodes of p,r,...,s, respectively, and let $c=(c_0,c_1,\ldots)$ be a computation of P with valuations in data domain D:

(i) if A is a classical first-order formula, then $D,c_j \models A$ in the usual sense when values of variables of P are given by c_j,

(ii) if A is of the form *at* n, then $D,c_j \models A$ iff c_j is of the form $<n_p n_r \ldots n_s, \ldots>$ where $n \in \{n_p, n_r, \ldots, n_s\}$,

(iii) $D \models \{A\}P\{B\}$ iff for any computation c of P such that $D,c_0 \models A$, $D,c_j \models B$ for any $j \in \omega$,

(iv) if F is a set of classical first-order formulae, then $F \models \{A\}P\{B\}$ if for any model D of F, $D \models \{A\}P\{B\}$.

4.3. Proof Rules

(I) $\vdash \{A\}p\{at\ init(p) \rightarrow A\}$,

(E) $\{A\}p\{at\ n \rightarrow B\} \vdash \{A\}p\{at\ m \rightarrow B\}$, where $n \approx_p m$,

(N) $\vdash \{A\}p\{at\ m \rightarrow G \wedge A\}$,
where p contains only a transition leading to m and executing G|skip,

(S) $\{A\}p\{at\ n \wedge G \rightarrow B[x_1/t_1,\ldots,x_k/t_k]\} \vdash \{A\}p\{at\ m \rightarrow B\}$,
where p contains only a transition leading from n to m and executing G | x1,....,xk:= t1,...,tk,

(MS) $\{A\}r\{at\ n \wedge G \rightarrow B[Q/insert(e,Q)]\} \vdash \{A\}r\{at\ m \rightarrow B\}$,
where Q is the variable representing the queue of messages sent to p, and r contains only a transition leading from n to m and executing G | send e to p,

(MR) $\{A\}p\{(at\ n \wedge G \wedge Q \neq empty) \rightarrow B[x/front(Q), Q/delete(Q)]\}$
$\vdash \{A\}p\{at\ m \rightarrow B\}$,
where Q is the variable representing the queue of messages sent to p, and p contains a single transition leading from n to m and executing G | receive x,

(R;)

(i) $\bigwedge_{n \in term(p)}\{A\}p\{at\ n \to B\},\ \{B\}r\{at\ m \to C\} \vdash \{A\}p;r\{at\ m \to C\}$,
where $m \in nodes(r)$.

(ii) $\{A\}p\{at\ k \to B\} \vdash \{A\}p;r\{at\ k \to B\}$, where $k \in nodes(p)$

(R+) $\{A\}p\{at\ n \to B\},\ \{A\}r\{at\ n \to B\} \vdash \{A\}p+r\{at\ n \to B\}$,

(R=)

$$\{A\}graph(E_p)\{at\ n \to B\}, \{A\}graph(E_p)\{\bigwedge_{k \approx_p init(p)} at\ k \to A\} \vdash \{A\}p\{at\ n \to B\}$$

where p is defined by equation $p = E_p(p)$,

(R||)

$\{A\}p\{B\},\ \{A\}r\{B\},...,\ \{A\}s\{B\} \vdash \{A\}p||r||...||s\{B\}$,

(G) $\{A\}p\{\bigwedge_{n \in nodes(p)} at\ n \to B\} \vdash \{A\}p\{B\}$,

(L)

(i) $B \to C,\ \{A\}p\{B\} \vdash \{A\}p\{C\}$,

(ii) $C \to A,\ \{A\}p\{B\} \vdash \{C\}p\{B\}$,

(iii) $\{A\}p\{B\},\ \{A\}p\{C\} \vdash \{A\}p\{B \wedge C\}$.

Let us now briefly discuss the soundness of the proof system. We want to show that for any set F of classical first-order formulae, $F \vdash \{A\}P\{B\}$ implies $F \models \{A\}P\{B\}$.

As usually, it suffices to show that for any model M of F, and any rule R of the proof system, if the premises of R are true in M, then also the conclusion of R is true in M.

The soundness of rules (I), (E), (N), (S), (R;), (R+), (G) and (L) should be obvious.

The soundness of (MS) and (MR) follows immediately from the definition of operational semantics, where we assumed that sending (receiving) a message depends on inserting (deleting) the message into (from) the respective queue.

The soundness of (R=) follows from the fact that all nodes in an annotated graph which are to be identified with the initial node of p have to satisfy the initial condition.

The soundness of (R||) follows from the fact that we assumed interleaving as the underlying model of concurrency. Namely, in the view of (G), the premises of (R||) assure that the formula B is preserved by any transition of p, r,..., s. Thus it is preserved by any sequence of transitions of p, r,..., s.

5. A SIMPLE EXAMPLE

Let us consider a simple example of two processes, each with an initial set of items, which communicate to arrange that one process has the smaller items, the other the larger ones.

In the figure below by Init and Inv we abbreviate the following formulae:

Init $\leftrightarrow$

$$\text{Small.Q}=\text{Large.Q}=\text{empty} \wedge S=S0 \wedge L=L0 \wedge$$
$$L0 \cap S0=\emptyset \wedge 0 \notin S0 \wedge 0 \notin L0 \wedge |S0|>0 \wedge |L0|>0$$

Inv $\leftrightarrow$

$$L \cap S = \emptyset \wedge S \cup L \subset S0 \cup L0 \wedge$$
$$\forall z \in \text{Large.Q}\ z \notin L \wedge z \notin S \wedge z \in S0 \cup L0 \wedge$$
$$\forall z \in \text{Small.Q}\ z \notin L \wedge z \notin S \wedge z \in S0 \cup L0 \wedge$$
$$\forall z1 \in \text{Small.Q}\ \forall z2 \in L\ z1<z2 \wedge$$
$$x<\min(L) \wedge x \notin \text{Small.Q} \wedge y \notin \text{Large.Q},$$

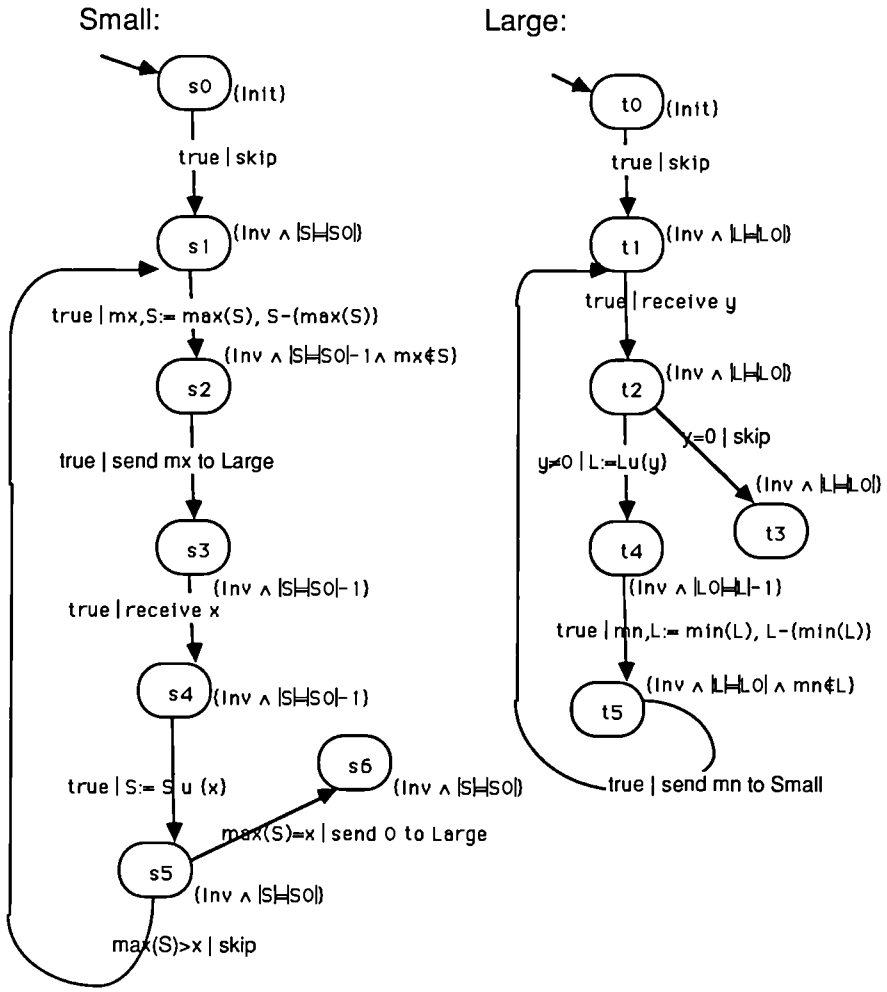
where x and y are local variables of Small and Large, respectively.

The program can be described e.g. by the following expression:

```
S = (s1',s2);(s2',s3);(s3',s4);(s4',s5); [(s5',s6) + [(s5",s7);S] ]
Small = (s0,s1); S
L = (t1',t2); [ [(t2',t3);(t3',t5);L] + (t2",t4)]
Large = (t0,t1); L
```

where (si,sj) and (tl,tm) denote respective transitions shown on the figure below.

Note that the figure illustrates the transition graph of the program as well as an annotated graph.



We are going to prove that for any formula A_i labeling a node n_i in Small||Large,

$$\{\text{Init}\}\ \text{Small} \parallel \text{Large}\ \{at\ n_i \rightarrow A_i\}.$$

First we show that

$$\{\text{Init}\}\ \text{Small} \parallel \text{Large}\ \{\text{Inv}\}.$$

By rule (R||) it suffices to show

$$\{\text{Init}\}\ \text{Small}\ \{\text{Inv}\}\ \text{and}\ \{\text{Init}\}\ \text{Large}\ \{\text{Inv}\}.$$

By rule (G) it reduces to proof that for any node s_i in Small (Large),

$$\{Init\}\ Small\ \{at s_i \rightarrow Inv\}.$$

Now the proof can be carried out by using provided rules.

In order to prove remaining formulae (e.g. {Init}Small{*at* s6 → |S|=|S0|}) one can apply respective rules (e.g. to prove {Init}Small{*at* s6 → |S|=|S0|} it suffices to show that {Init}Small{*at* s5 → (Inv∧|S|=|S0|)} and to apply rule (MS)).

Note that from the above we can deduce that
{Init} Small || Large
{*at* s6∧*at* t3 → (L∪S=L0∪S0 ∧ L∩S=Ø ∧ ∀z1∈S∀z2∈L z1<z2)}.

6. CONCLUSIONS AND RELATED WORK

Many authors gave proof systems that enable us to prove properties of concurrent processes (cf. e.g. [1,7,9] or the surveys [2,9]). A proof system given in [9] is maybe the one most closely related to that given in our paper. Misra and Chandy present a proof system for networks of processes communicating by message passing. They specify processes by pairs of assertions. The notation r|h|s means that s holds initially in process h and the holding of r at all times prior to some message implies that s holds at all times prior to and following that communication. We think, however, that the pairs of assertions we use are more natural and easier to use. In fact, the assertions we use generalize the well-known assertions of Hoare [5]. Moreover, in [9] the authors model networks of processes using CSP-like notation. The model of processes we have investigated is fully asynchronous and needs no CSP-like (thus synchronous) primitives.

Our work has to be seen as an ongoing development of formal methods for asynchronous systems in the FORMAST project.

REFERENCES

[1] Apt, K.R., Francez, N. and de Roever, W.P., A Proof System for Communicating Sequential Processes, ACM TOPLAS, 2, 3, (1980), 359-385.

[2] Barringer, H. A Survey of Verification Techniques for Parallel Programs (LNCS 191, Springer-Verlag, 1985)

[3] Dijkstra, E.W., A Discipline of Programming (Prentice-Hall Inc., 1976)

[4] Hemdal, G.A.H. and Coombs, C., Softchip Technology: A New System Architecture for Telecommunication Processing and Other Real Time Systems, 6th Int. SETSS Conference, Eindhoven,1986 (IEE Publication)

[5] Hoare, C.A.R., An Axiomatic Basis for Computer Programming, Communications ACM, 12, (1969), 576-580.

[6] Hoare, C.A.R., Communicating Sequential Processes, Communications ACM, 21, 8, (1978), 666-667.

[7] Levin, G.M. and Gries, D, A Proof Technique for Communicating Sequential Processes, Acta Informatica, 15, (1981), 281-302.

[8] Milner, R, A Calculus of Communicating Systems (LNCS 92, Springer-Verlag, 1980)

[9] Misra, J. and Chandy, K.M., Proofs of Networks of Processes, IEEE TOSE, SE-7, 4, (1981), 417-426.

[10] de Roever, W.P., The Quest for Compositionality: A Survey of Assertion Based Proof Systems for Concurrent Programs: Part 1: Concurrency Based on Shared Variables, Proc IFIP Work. Conf. on the Role of Abstract Models in in Information Processing, E.J. Neuhold and G. Chroust (eds.), (North-Holland, 1985), 181-206.

III. ADVANCED INFORMATION PROCESSING

Paper presented in the Plenary Sessions:

Parallel Sessions

Project No. 1085

PHASE 2 OF THE RECONFIGURABLE TRANSPUTER PROJECT - P1085

J G Harp
Royal Signals and Radar Establishment
St Andrews Road, Malvern, Worcs. England

Abstract

ESPRIT project P1085 has the objective of developing a high performance multiprocessor computer with supporting software and a range of applications to demonstrate its performance. This paper discusses the progress made at the mid-point of the project in meeting this objective. Prototype machines based on a reconfigurable architecture have been built and are described. Software tools have been defined and are being implemented. Applications have been developed using Occam and the high performance being achieved is presented.

1. INTRODUCTION

Assuming that biological systems got it right a few million years ago with parallel processing, then highly parallel Multiple Instruction Multiple Data (MIMD) architectures appear to have significant advantages over other architectures. MIMD machines have the potential of high performance with the flexibility to suit a wide range of applications but with the known difficulties in control and programming. ESPRIT project P1085 has the objective of developing a MIMD multiprocessor machine with supporting software to demonstrate that high performance can be achieved over a wide range of applications.

Project P1085 is seeking a low cost solution to the high computational demands of scientific and engineering applications by developing a VLSI processor which will be replicated to form a highly parallel computer. The processor being developed is a second generation transputer with on-chip floating point processor. High performance is being achieved by using large numbers of transputers. In order to optimise communications for the widest range of applications, the philosophy of the P1085 architecture is based on a reconfigurable interconnection stategy to give a Reconfigurable Transputer Processor. The machine is modular based on nodes of transputers with the interconnections (or links) between transputers being via software controlled switches. Single nodes of typically 18 transputers can be used as powerful workstations or nodes may be connected together, again reconfigurably, to give machines with supercomputer performance.

Operating systems, high level languages and support software are being developed and implemented to ease the programming burden and software to allow both hardware and software to be debugged is in an advanced state of development.

Significant progress is being made in demonstrating that high performance can be achieved in applications ranging from the high data rates of signal processing, through pattern recognition to general applications in science and engineering.

[This work is partially supported by CEC under contract P1085]

In the following sections we describe the progress at the mid-point of the project in developing hardware, software and applications and the early results being achieved.

2. HARDWARE

P1085 is developing a modular, hierarchical architecture based on reconfigurable nodes of transputers. Network topology, both within and between nodes, is determined by VLSI switches controlled by transputers; thus the topology of the architecture can be optimised for any specific application. This architecture allows the communication between processors to be configured to match the bandwidth requirement of the application. Additionally, the topology can be configured to match the algorithm reducing the programming burden. A more detailed philosophy of the architecture is given in [1].

Each node (or supernode as it is called) consists of 16 worker transputers as shown schematically in figure 1. Each worker transputer will be a T800 device which integrates a 32 bit CPU, a 64 bit floating point unit, four standard transputer communication links, 4Kbytes of RAM, a memory interface and peripheral interface on a single chip fabricated using a 1.5 micron CMOS process.

The four links of each worker transputer are connected to a 72x72 VLSI switch which is controlled by a further transputer which also has its links connected to the switch. The 72x72 switch is implemented in two NEC ICs, each functionally equivalent to a 72x36 cross-bar switch. Each T800 has 256 Kbytes of external memory and the node has an additional transputer with a large amount of memory (16 Mbytes) which can be used for storing and distributing data and code. There is an option of incorporating a winchester disk, controlled by a M212 transputer into the node.

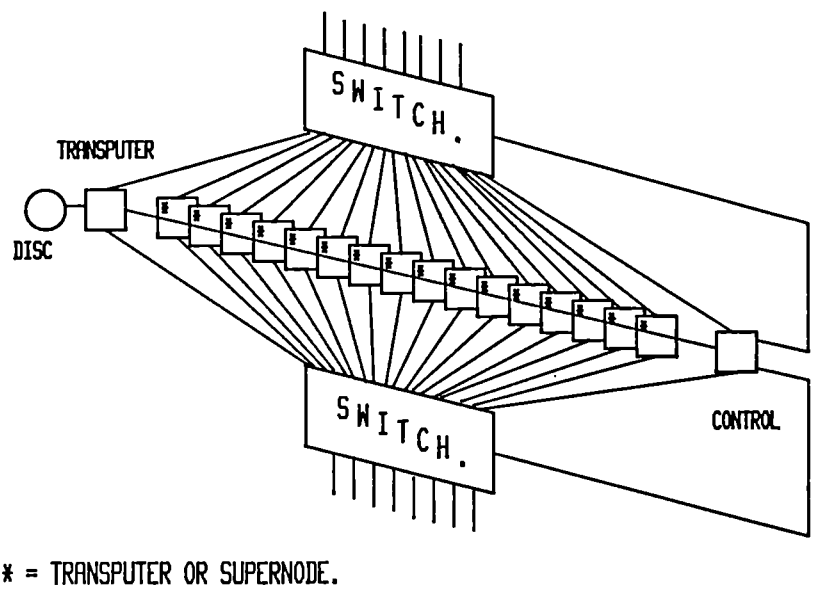


Figure 1 RTP Schematic

A program running on the control transputer can set up any interconection network, in a re-arrangable non-blocking mode, for all links within the node. Thus within the node, using one 72x72 switch, any pair of links can be connected. For machines consisting of more than one node, two 72x72 switches are used to allow any link to be connected outside the node. An additional internode switch is used to implement a 3-stage Clos network for reconfiguration between nodes. Thus in figure 1, the components designated as worker transputers can equally well represent nodes, giving a hierarchical machine. The ability to implement any 4-connected network over the whole machine frees the programmer from having to consider the node structure, though he may wish to do so in order to simplify programming.

An important feature within the node is a control bus which enables any transputer to communicate with the control transputer independently of the links. This facility has been incorporated for synchronisation and for debugging programs. Again, the control bus is hierarchical allowing the control transputers to communicate with each other and with a host computer.

A computer with the power of RTP can 'consume' or generate vast amounts of data and we are developing a Fast I/O node to enable high speed input and output of data. The Fast I/O node (figure 2) consists of essentially, sequencers and link adaptors. Data is fed to or from the peripherals (radar, TV camera or imager) into sequencers implemented as FIFO memories controlled by a chain of programmable counters offering a flexible method of segmenting and routing the data. The data is connected to the RTP through 64 link adapters via a transputer controlled switch giving a consistent system architecture. Data can be routed from the fast I/O node to any processors in the machine and from the programmers viewpoint the Fast I/O node appears as any other node in the system.

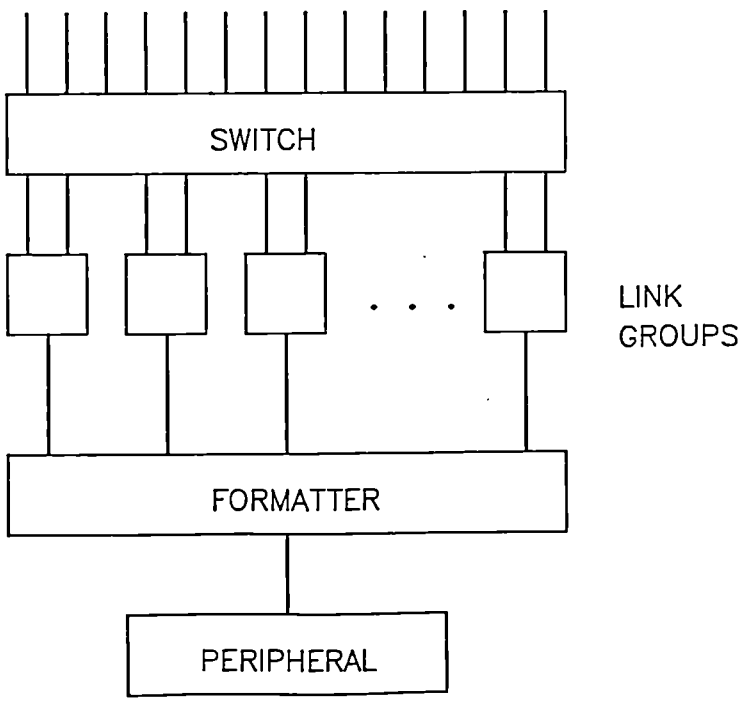


Figure 2 Fast I/O Node Schematic

In general there are three modes of operation of any switched system; static, quasi-static and dynamic. In static switching, programs are written for a fixed topology that matches the algorithm. The program is compiled and configured producing a configuration table. The switches are set using data from the configuration table and the program is down loaded and executed. Static mode switching may be used to partition the machine into independant units to allow simultaneous access for multiple users.

In quasi-static switching, the machine configuration can be changed at pre-determined synchronisation points. For example, in image processing, the topology is first configured to allow the image to be input at maximum data rate. Once data is distributed over the processors, the machine may be configured to a two-dimensional array (or mesh) for low level image processing operations such as convolution and segmentation. After completion of low level operations, all communication ceases and the the machine is configured into a tree, say, for high level pattern recognition.

In dynamic switching, any link can be connected to any other link at any time provided that there is no communication on the links being switched. Dynamic switching allows dynamic load balancing and fault tolerance and offers potential benefits for efficient implementation of some high level languages.

The RTP architecture supports all three switching modes and is currently being used statically and quasi-statically. There are potential problems in dynamic switching such as bottlenecks in the switch controller, overheads in synchronisation and software control. Full exploitation of dynamic switching is beyond the current resources of P1085 and is not being pursued in detail.

3 SOFTWARE

If the architecture is successful then the software has to support a wide spectrum of users from the sophisticated programmer who wishes to obtain maximum performance from the machine to the naive user with little experience of parallel architectures and who does not necessarily wish to know details of the hardware. Here we have a conflict in that we are making a general purpose machine for a range of applications varying from those that require maximum performance at the expense of increased programmer effort, to users with limited expertise but who are willing to accept lower hardware performance and efficiency. We are attempting to satisfy the spectrum spanned by these extremes by developing software based on the Occam model of computation. Within this model, hardware features such as the switch and control bus can be hidden from the naive user

The RTP software is based on a three layer model. Layer 1 is the system utility software and includes occam-2 and a modified Transputer Development System. The basic modules are a system environment package which maintains a small data base of system components and systems status (eg number and types of nodes, memory and switch setting) and an interface between the nodes and a host. There are modules for controlling the switch and the bus, hardware checking and error handlers.

Layer 2 software is the higher level support and operating system tools which will extract communication graphs from an occam program, extract switch settings from the graphs, and configure the machine topology. Research topics in layer 2 include automatic allocation of processes to processors and the control of process migration. An advanced communications kernel will allow the programmer the freedom to use as many links per processor as he requires with links greater than four being mapped onto soft channels using a message switching protocol.

The layer 3 software is the high level programming stage and includes parallel Prolog, and real-time and set theoretic languages.

4 APPLICATIONS

Applications are being developed in the fields of signal and image processing, image synthesis, science and engineering, computer aided design and computer aided manufacture. The applications are chosen to test the power and flexibility of the RTP architecture. They vary in data throughput requirements, data storage requirement, number crunching demand and interfaces with both peripherals and operators. For example, signal processing demands regular operations on high bandwidth structured data whereas the emphasis in scientific applications (such as lattice gauge theory) is on high accuracy floating point computation. The CAD application is concentrating on logic simulation of VLSI circuits and demands a good user interface and large amounts of memory. In the CAM application, the RTP architecture has to be interfaced to local area networks.

5 RESULTS AND ACHIEVEMENTS

After 18 months of the 3 year project we have made progress in architecture development, software and applications.

5.1 Hardware

A key component of the architecture is the switch circuit. This component has been designed and fabricated on a 14000 gate-array IC. NEC have supplied the first 50 engineering samples which have been tested and shown to meet the specification. Prototype supernodes, RTP-1 and RTP-2, incorporating switch ICs have been constructed and used by the applications programmers. The RTP-1 version (shown in figure 3) is implemented as 5 circuit boards and consists of a transputer controlled frame store, two boards each containing 8 transputers (T414s) with each transputer having 128 Kbytes of external RAM, a switch and controller board and a display board. RTP-2 is similar but has 16 transputers each with 256 Kbytes of RAM on 4 PCBs. RTP-1 and RTP-2 do not have a control bus.

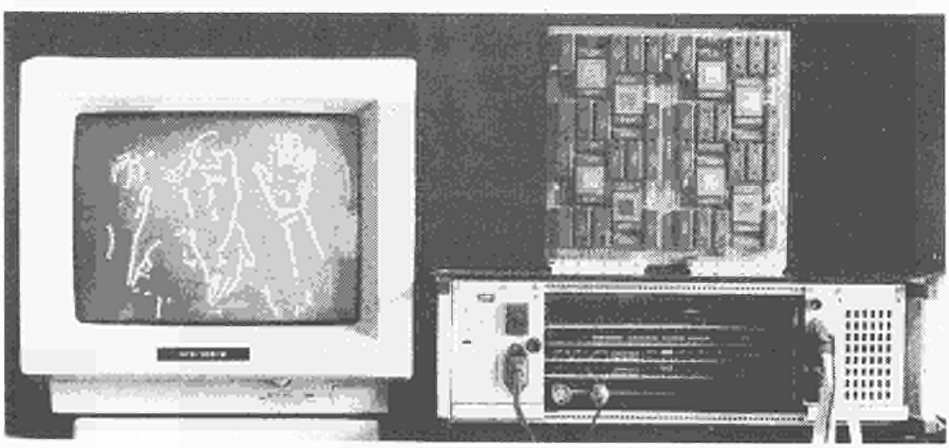


Figure 3 RTP-1

First samples of the T800 have been supplied to the consortium and the following performance figures have been achieved :

operation	single length	double length
add	350 ns	350 ns
subtract	350 ns	350 ns
multiply	650 ns	1050 ns
divide	950 ns	1700 ns

A more realistic measure of performance is the Whetstone benchmark and the performance of the T800 is compared with other processors below:

processor		Whetstones/sec single length
Intel 80286/80287	8 MHz	300K
Inmos T414-20		663K
NS 32332-32081	15 MHz	728K
MC 68020/68881	16/12 MHz	755K
ATT 320000/32100		1000K
Fairchild clipper	33 MHz	2220K
T800-20		4000K

The T800 has more memory and higher link speed than the T414 transputers. On-chip memory is increased to 4 Kbytes and an overlapped acknowledgement protocol achieves a data rate of 1.8 Mbytes per second in one direction on a link, or 2.4 Mbytes per second overall rate when the link carries data in both directions simultaneously. Graphics support is provided in the T800 by the incorporation of a two-dimensional block move.

Initial samples of the T800 had bugs and "preventative programming" had to be used. Commercial devices are expected by September and it is intended to demonstrate applications running on RTP with T800 transputers at this conference.

5.2 Software

A test and exerciser harness has been written and is being used to develop system utility modules. Initial versions of the switch and control bus modules have been written. A major component of the baseline software is the systems environment package whose functionality and interfaces are being refined. A family of operating system tools have been defined during the study phase indicating that it is possible to ease the programmers task. A file server has been implemented to interface the transputer resident development system to a host running under Unix.

As a first step in developing a Prolog compiler, a macro assembler (AST) has been written for the transputer. AST has been used to produce a Prolog compiler and a single transputer version of Prolog is now being tested and documented.

No matter how good the operating system, program development on a parallel machine is more difficult than on a serial machine. When debugging a crashed program which is distributed over a number of processors the programmer has to to be able to access information stored in each processing element and then analyse that information. Task 12 is developing an interactive debugging system. Two methods of retrieving information are being developed; one based on an Analyse worm and one based on the control bus. The

analyse worm is essentially based on a bootstrapping program which can relay information on the state of any processor via links to the host. Using the control bus has the advantage of being able to access deadlocked transputers as it is independent of the links.

A pilot post-mortem debug system based on the analyse worm has been produced and used successfully in a restricted hardware environment on a number of occam programs. Following a program crash it is possible to determine the the state of execution within the occam source on individual transputers. The user interface will show the current occam instruction on halting, access selected variables, trace through procedures and access the hardware state. In the next phase an advanced user interface will be developed together with software to access the state of the array via the control bus and tools to allow 'breakpoint and continue'.

5.3 Applications

All six applications work packages have completed a study phase, developed representative algorithms and implemented those algorithms on arrays of transputers. In every case, parallelism has been identified and exploited.

From the applications, a methodology is being developed to aid in distributing programs onto parallel architectures [2]. We use three broad classes of parallelism:

i) Event Parallelism where each processor executes the same program on a different, but complete, data set and where only a minimum amount of communication is necessary.
ii) Geometric Parallelism where each processor executes a similar program but on a subset of the data. Communication is increased as each processor may require access to data stored in other processors.
iii) Algorithmic Parallelism where each processor executes a segment of the algorithm and where the data tends to flow through pipelines of processors.

Algorithmic parallelism appears a natural mapping but care is required in determining control and communication; it is very easy to get it very wrong with consequent low efficiencies. Geometric parallelism is generally easy to code and tends to give high efficiency but requires more memory to store the replicated code.

We can use this classification as an aid to identifying parallelism in an algorithm and for configuring the topology to optimise efficiency. Applying these methods to the solution of Laplace's equation gives the following timings:

method	time	efficiency	Mean memory/processor
Sequential on 1 T414	143 ms	-	15K
Geometric on 4 T414s	37 ms	96%	7K
Algorithmic on 4 T414s	64 ms	56%	5K
Hybrid on 16 T414s	17 ms	52%	2K

The hybrid approach combines the advantages of geometric and algorithmic methods, and in practice some form of hybrid approach will probably be used on large multi-node machines. The sequential algorithm was implemented in occam on a 68000 based machine and executed in 2.57s.

Algorithms for logic simulation for VLSI circuits being developed in the CAD application can exploit a combination of event and algorithmic parallelism. Here we are developing a logic simulator called Lucky-Log which uses an event-driven simulation [3]. Lucky-Log is based on a set of interconnected units where each unit consists of three tranputers; two Event Computing Transputers and an event manager transputer. Each unit operates on

an event list and comunicates the results to a display manager transputer. Thus each supernode can implement 5 simulation units and a display manager.

Luck-Log is being implemented on T414 transputers and the following results have been measured and predicted for T800 transputers:

3 x (T414 + 2 Mbyte DRAM)	5000 EPS
3 x (T800 + 2 Mbyte DRAM)	10000 EPS
Supernode	75000 EPS

where 1 EPS is defined as search the event list, compute all gate outputs and then update the event list. (HILO on a VAX780 runs at 1500 EPS).

Algorithms for low level image processing map naturally in a geometric fashion onto two-dimensional arrays of transputers. Each processor operates on a rectangular sub-image and communicates overlapping regions to each of its neighbours after each operation. We have implemented a range of functions for filtering, convolution, segmentation, histogramming etc.[4]. We have demonstrated that 16 transputers can filter, edge detect and segment a 128x128 image in 80ms and that performance improves linearly as the number of processors is increased.

Space limitations do not allow a description of all P1085 applications and the results described above are typical.

6. CONCLUSIONS

At the mid point of the project we have defined an architecture and built prototype machines. Initial samples of the T800 floating point transputer have been supplied to the project and a performance 3 times higher than the original target specification has been achieved.

Much of the software activity began as research topics. We have identified operating system tools and methodologies to improve programmer productivity and some of these tools will be implemented in the next phase. The work on debugging techniques has been particularly successful and has produced software which is now being incorporated into a commercial product. However, software systems for highly parallel architectures are still in their infancy and much more research is required.

A range of applications have been programmed in Occam and run on networks of transputers. High efficiencies have been measured demonstrating that it is possible to program MIMD machines and that high performance can be achieved in diverse applications. An initial, but important, conclusion is that reconfiguration makes the machine easier to program.

During the remainder of the project machines will be manufactured and supplied to the partners for applications development and for software research. It remains to be seen if the encouraging performance that has been attained on a few tens of processors can be achieved on a few hundred processors.

7. ACKNOWLEDGEMENTS

The work reported on here has been performed by the P1085 consortium as a whole and the author is merely a mouthpiece. The results have been achieved by a lot of dedicated and intensive work by experts too numerous to mention. The author gratefully acknowledges all members of the project for their contributions.

8. REFERENCES

1. Harp JG, Jesshope CR, Muntean T, Whitby-Strevens C.
"Phase 1 of the development and application of a low cost high performance multiprocessor machine".
ESPRIT 86:Results and Achievements, Elsevier Science, 1987, 551-562
2. Pritchard DJ, Askew CR, Carpenter DB, Glendinning I,
Hey AJG, Nicole DN. : "Practical Parallelism using Transputer Arrays"
Proc PARLE Conf., Eindoven, 1987
3. Werner J, Beresford R
"A system engineers guide to simulators"
VLSI Design, Feb 1984, 27-31
4. Harp JG, Palmer KJ, Webber HC
"Image Processing on the Reconfigurable Transputer Processor"
Proc 8th OUG Technical Meeting, To be published.

9. P1085 PARTNERS

Royal Signals and Radar Establishment
St. Andrews Rd, Malvern, Worcs. UK

Apsis
Chemin du Vieux-Chene, Zirst 38240, France

Inmos Ltd
1000 Aztec West, Almondsbury, Bristol, UK

Laboratoire Genie Informatique (IMAG)
University of Grenoble, BF68, 34402 Grenoble, France

Telmat SA
ZI Route s'Issenheim, F-68360 Soultz, France

The University
Southampton, UK

Thorn-EMI Central Research Labs
Dawley Road, Hayes, Middlesex, UK

Project No. 530

STATUS AND EVOLUTION OF THE EPSILON SYSTEM

Giorgio Levi*, Mario Modesti+, Jacques Kouloumdjian§

* Dipartimento di Informatica
Università di Pisa - Corso Italia, 40-56100 Pisa - Italy
Tel. +39 50 510246

+ Systems & Management SpA
Vicolo S.Pierino, 4-56100 Pisa - Italy
Tel. +39 50 598084 (598035), Tx. 320629 SEM MI

§ Universite` Claude Bernard de Lyon
Laboratoire informatique IUT 1
43, Bd du 11 nov 1918, 69622 Villeurbanne CEDEX
Tel: 78 94 88 57

1. The EPSILON prototype

Epsilon /Coscia86,87/ is a prototype of a knowledge base management system developed within ESPRIT Project 530. Epsilon is built on top of commercial PROLOG and Relational Data Base Management systems, running on standard UNIX environments. The main concepts underlying the Epsilon approach are:

1. the extension of PROLOG with theories (multiple worlds),
2. the definition of a transparent interface from PROLOG to Relational Data Base Management Systems
3. the use of metaprogramming as the basic technique to define new inference engines and tools,
4. the use of partial evaluation techniques as a systematic method to "compile" metaprograms,
5. the definition of a graphical user interface on a personal computer.

This paper gives an overview of the current status concerning points 1, 3 and 4.

Data bases can be integrated in the KBMS according to an interpretive or a compilative approach. In the first case, the PROLOG interpreter interacts with a component, which translates conjunctions of edb-predicates into complex queries to the DBMS. In the compilative approach /Coscia86/, logic programs are first translated into relational algebra expressions and then further optimized. We are only interested here in stressing that Data Bases are viewed by the user through the uniform theory mechanism.

The user interface is defined on a Macintosh, which takes care of all the graphical operations without affecting the performance of the host UNIX machine and provides user-friendly features (such as windows and menus) and nice knowledge editing facilities.

2. Adding theories to PROLOG

The **theory** is the basic component of the Epsilon knowledge base. Theories are similar to worlds in MULTILOG /Kauffmann86a,86b/ and to unit worlds and instances in MANDALA /Furukawa84/.

Namely, they are composed of a chunk of knowledge, associated to a specific inference machine (theory processor).

A theory corresponds to a chunk of knowledge contained in a file and is associated to a window in the graphical user interface. The theory processor contains operations to query the theory, to update and search the theory, to load/unload the theory, considered as an atomic separate object. The theory processor can also contain tools (debugger, tracer, explanator, query-the-user). Epsilon provides two primitive theory processors (or classes): the first one handles the language PROLOG extended with the theory feature, while the second one handles Data Base theories. As we will discuss later, new classes (theory processors) can be defined.

Adding theories to standard PROLOG allows to define a structure on the PROLOG workspace and provides a renaming mechanism which guarantees that each theory contains objects whose names differ from object names occurring in other theories. This mechanism is currently simulated on top of a commercial PROLOG compiler.

The kernel of Epsilon maintains a Knowledge Base Dictionary, which contains a description of the existing theories (in particular, their classes). Theories can communicate by making a reference to the generic operations for querying and updating theories. The kernel uses the Knowledge Base Dictionary to select the operations of the proper inference engine.

Let us consider the simple example in Figure 1, where the theory **dbrules** has class Prolog and the theory **stores** has class Data Base, as shown by the Knowledge Base Dictionary (**KBinfo**). The window corresponding to the Data Base theory does not contain the Data Base tuples, but contains the (PROLOG representation of the) Data Base Dictionary. The PROLOG relations defined in **dbrules** are essentially views on the data base, defined by an explicit reference to the query operation **call(Goal,Theory)**. The kernel will translate the generic invocation of **call** into an invocation of the specific query operation in the Data Base theory processor.

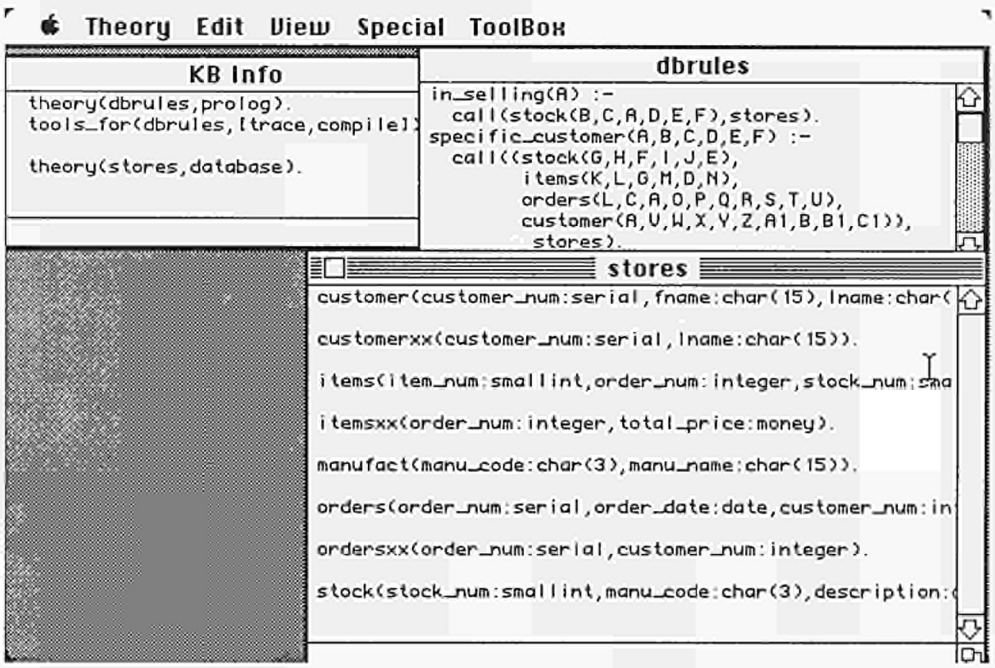


Figure1. Two theories and the Dictionary.

3. Defining new classes (theory processors) by metaprograms

Inference engines are handled as first-class citizens in Epsilon, since new inference engines

can be defined inside theories. A knowledge base is then composed by homogeneous objects (theories) that can indifferently be either user (object level) theories or theory processors for other theories. If a theory T has class C, there exists a theory named C containing the inference engine of T. It is therefore possible to build in a cleaner and natural way knowledge bases relying on specific domain knowledge and multiple layers of general (control) knowledge, and to extend in a simple and efficient way the features of the system without modifying the kernel.

A theory defining an inference engine for a class of theories must define the programs for querying (call) and for updating (assert and retract). Moreover, an engine can define tools.

Metaprogramming is used to define the various inference machines. The power of metaprogrogramming in logic languages is shown by the structure of the well-known three-lines metainterpreter /Sterling84/.

```
1. solve(true) :- !.
2. solve((Q1,Q2)) :- !, solve(Q1), solve(Q2).
3. solve(Q) :- clause(Q,Body), solve(Body).
```

The definition is so simple and short, because most of the job is done by the underlying "true" object level interpreter. The metainterpreter does not explicitely define the two computationally most complex procedures, i.e. backtracking and unification. Backtracking is, in fact, implicit in clause 3, since the atom clause(Q,Body) is backtrackable. Unification, on the other hand, corresponds directly to the object level unification.

The definition of "enhanced" metainterpreters becomes attractive, because it allows to define new functionalities without modifying the program (the object level knowledge) and the basic interpreter. Enhanced metainterpreters can embed new control strategies, extend the logic language with new useful constructs (for instance, knowledge structure, or uncertainties) and the related inference rules (inheritance or approximate reasoning), or define analysis tools, to provide typical expert systems (explanation, query-the-user, etc.), or interactive monitoring (debugger, tracer, etc.) capabilities.

One of the main features of the metaprogramming approach is its ability to extend the language, the inference machine and the environment, without modifying the basic building blocks, i.e. the PROLOG interpreter and compiler. The extensions defined as (PROLOG) metaprograms are easy to define and portable, as is the case for tools in LISP environments. Their performance is anyway rather poor, if compared to what could be obtained by an ad-hoc implementation of the new language/environment, which, however, is a very expensive solution, and, in addition, is not necessarily open to further extensions and modifications. Metaprogramming is, instead, easy, more flexible and clean, since the knowledge (the rules in the possibly extended language) and the inference engine (the metainterpreter) are separate and easy to understand, and all the extensions in the inference engine are clearly defined at the meta-level.

The real drawback of metaprogramming is performance. There exists, however, an interesting technique (partial evaluation of metainterpreters), which allows to combine the low cost and the high flexibility of metaprogramming with performance. This technique will be discussed in Section 7.

4. Defining new features by links in the Knowledge Base

A new inference engine conceptually defines a new knowledge representation language. The new language features can either affect the object level description language (as is the case, for instance, of clauses extended with uncertainties and of PROLOG extended with coroutining) or be represented at the meta-level, as relations among theories. We will mainly discuss the last case, which is realized in Epsilon defining **links** between theories and by representing them in the Knowledge Base Dictionary.

Some links define "new" inference rules for a theory. In such a case, the inference rule must be embedded in the query metainterpreters of the theory processors. We will discuss an example in the next section.

Other links define constraints on the theory updating and are supported by those components of the inference engine which handle the theory updating. This is the case of the **constraint** link. If a theory T_1 has its integrity constraints in theory T_2, any update operation on T_1 will cause an

invocation of a suitable component in the inference engine of T_2, which supports the language used to express the integrity constraints.

Other links are finally related to the Knowledge Base Management. This is the case of the primitive **version** link, directly related to the tools for program transformation and optimization provided by Epsilon. Examples are the partial evaluator of metaprograms and the transformation tools used in data base access optimization. The application of a transformation tool P to a theory T generates a derived theory T', which is linked by a **version** link to T.

Consider the example shown in Figure 2, where T3 is obtained by partially evaluating the knowledge in T1, under the metainterpreter demo, belonging to the inference engine of T1. T3 is linked to T1 by the link **version(demo)**. The semantics of such a link is twofold. When a call of demo applied to some predicate in T1 is found, it is converted into the corresponding call in T3. Moreover, when T1 is updated, T3 must be kept consistent with T1 updatings.

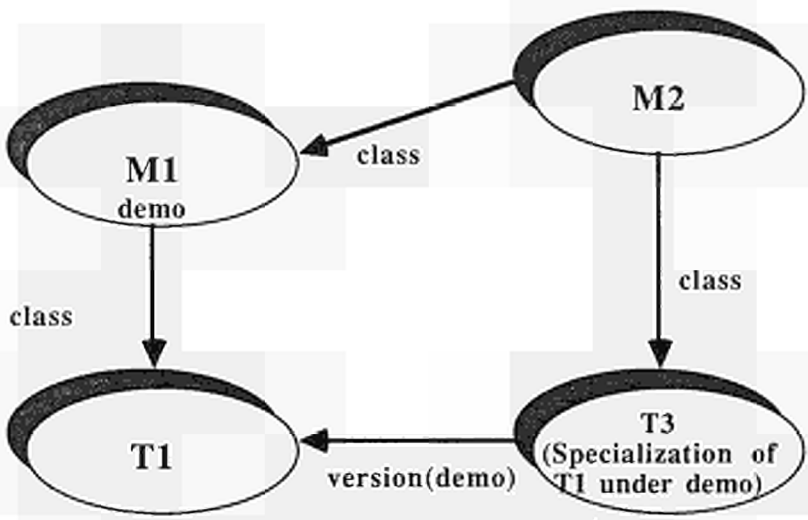


Figure 2. The version link in a partial evaluation of metaprograms.

In the current Epsilon prototype we have defined some non primitive classes. The most interesting is a class which defines various inheritance mechanisms, based on the definition of links between theories in the Knowledge Base Dictionary.

5. Inheritance rules and inheritance links

Default communication mechanisms between theories T_1 and T_2 are achieved by defining an inheritance link from T_2 to T_1. This link is interpreted by the query handler of T_1 as follows. If a subgoal cannot be solved in T_1, it is solved in T_2. Multiple inheritance is possible. In the case shown in Figure 3 (where the two theories share the same inference engine), the result is inheritance of the "object level" knowledge (i.e., the clauses of T_2 are available in T_1). The case in Figure 4, where the subgoal is solved by the inference engine of T_2, is typical of the interaction between theories having different inference engines.

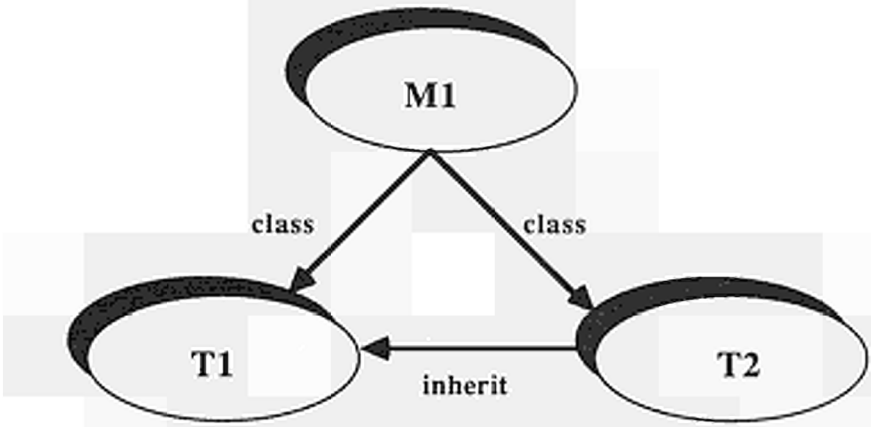


Figure 3. Inheritance of "object level" knowledge.

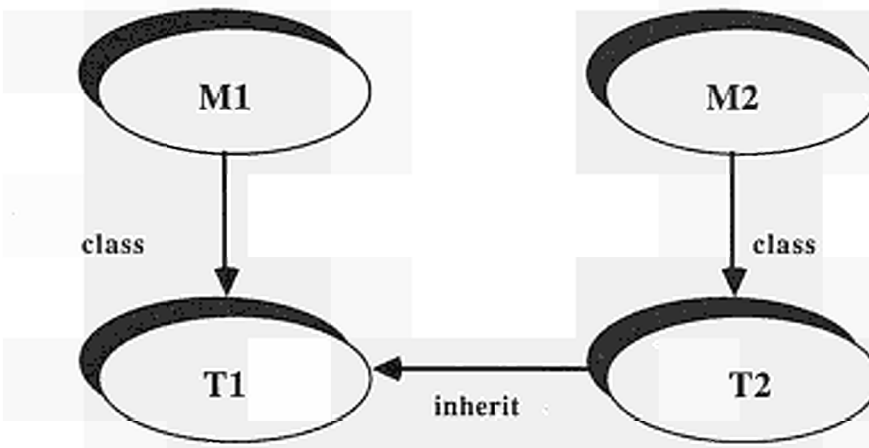


Figure 4. Inheritance through a different inference engine.

As an example, we can consider the case where T_1 is a logic program and T_2 is the set of tuples in a database. The database is visible to the logic program, through a specific inference engine, i.e. the data base query processor.

6. An example with links: The 3 wisemen problem

The inference engine used in the example (the theory **engine**) contains the metaprogram (query-metainterpreter) **scall** (in Figure 5). **scall** defines a language which is essentially PROLOG (including cut), extended with the above discussed inheritance rules,based on the links **clsinher** and **clsisa**. If T1 inherits from T2 through the link **clsinher**, then queries failed in T1 can be solved in T2. If T1 inherits from T2 through the link **clsisa**, then queries failed in T1 can be solved (in T1) using clauses of T2. The language used in all the examples is the PROLOG variant used in our project, where PROLOG primitives have generally the prefix "sm".

engine

```
scall(A,B) :- proquery(A,B).
proquery(!,A) :- !.
proquery((A,B),C) :- !, solve_body((A,B),D,E,C),
        (smidentical(D,cut), !, proquery(E,C)
        ; smsucceed).
proquery(A,B) :- smogsyspred(A), !, smcall(A).
proquery(A,B) :- smclause(A,C,B),solve_body(C,D,E,B),
        (smidentical(D,cut), !, proquery(E,B)
        ; smsucceed).
proquery(A,B) :-followlink(B,C,A),
        smclause(A,D,C), proquery(D,B).
proquery(A,B) :- existlink(B,C,clsinher),
        smfunctor(A,D,E), smnot(smthpreds(B,D,E)),
        scall(A,C).
solve_body(!,cut,smsucceed,A) :- !.
solve_body((!,A),cut,A,B) :- !.
solve_body((A,B),C,D,E) :- !,proquery(A,E),
        solve_body(B,C,D,E).
solve_body(A,nocut,smsucceed,B) :- proquery(A,B).
followlink(A,B,C) :- existlink(A,B,clsisa),
        smnot(smclause(C,D,A)).
followlink(A,B,C) :- existlink(A,D,clsisa),
        smnot(smclause(C,E,A)),followlink(D,B,C).
```

Figure 5. The query metainterpreter of the inference engine **engine**.

The problem. Each wiseman has a hat. At least one hat is white. Each wiseman can see the other wisemen hat and, when asked if he knows whether his own hat is white, will answer either yes or no. Wisemen are supposed to be perfect reasoners, i.e. the knowledge about the answer of a wiseman can be used by the other wisemen.

The structure of our solution is shown in Figure 6. The general knowledge, shared by all the wisemen, is in the theory **wiseman**. The specific knowledge of each wiseman is contained in a **view** theory. Each wiseman is represented by a theory which inherits the knowledge in **wiseman** (link **clsisa**) and in the corresponding **view** (link **clsinher**). Each wiseman can also communicate knowledge to the other wisemen views.

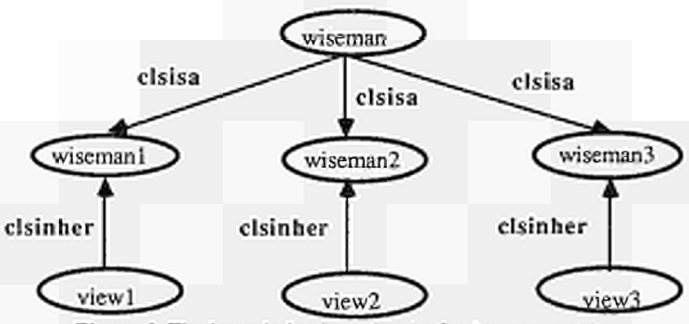


Figure 6. The knowledge base for the 3 wisemen problem.

The knowledge in the theory **wiseman** is shown in Figure 7. The theory knows about the 3 wisemen (next), their views (view), and contains essentially two rules. The first one (defining white) is the decision procedure in the yes case (i.e., the other hats are not white). The second rule communicates the don't know answer to the other wisemen, by adding new rules in their views.

The new rules are derived from the failure of currently existing rules.

Consider now the initial state shown in Figure 8, where all the hats are white. Each wiseman can be asked by issuing the query **answer(X)** in the corresponding theory. For example, the query for asking wiseman1, wiseman2 and wiseman3 one after the other, is

scall(answer(X),wiseman1),scall(answer(Y),wiseman2),
scall(answer(Z),wiseman3).

that will return X=no, Y=no, Z=yes. The resulting state for the theories is that shown in Figure 9, where the **view** theories contain also the new rules, inferred from the failures.

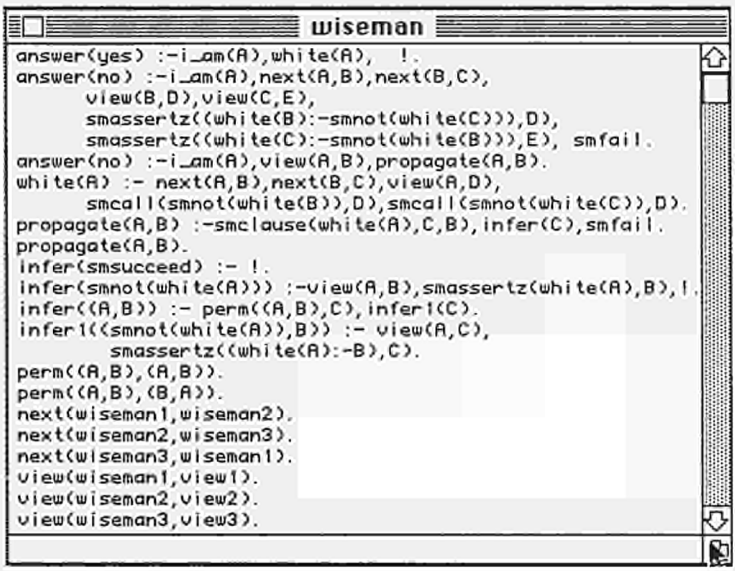


Figure 7. The theory wiseman.

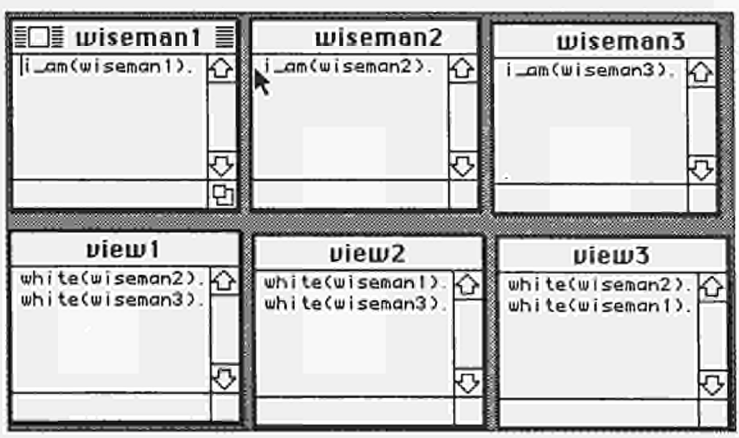


Figure 8. An initial state for the 3 wisemen problem.

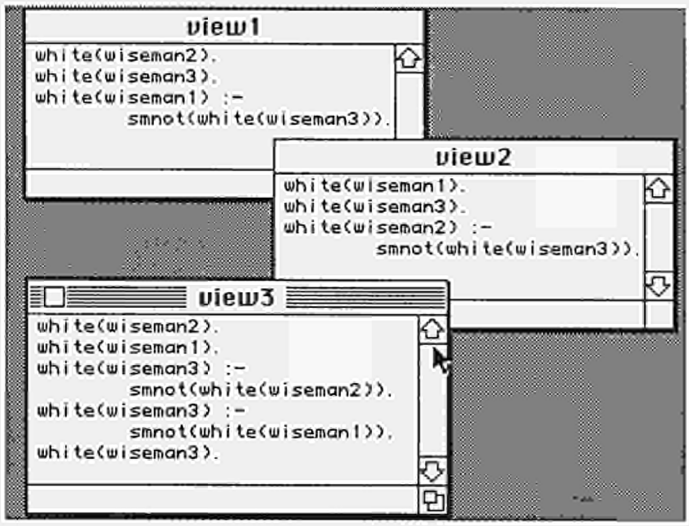


Figure 9. The final state for the 3 wisemen problem.

7. Metaprogramming and partial evaluation

Partial evaluation is based on Kleene's S-m-n Theorem: Given a function $f = \lambda x_1 \ldots x_n.e$ and k ($k \leq n$) specific (constant) values $a_1,\ldots,a_k$, we can effectively compute a function $f' = \lambda x_{k+1},\ldots,x_n.e'$, such that

$$f(a_1,\ldots,a_k,x_{k+1},\ldots,x_n) = f'(x_{k+1},\ldots,x_n).$$

f' is a specialization of f, hopefully more efficient than f for those specific input values. The essential aspects of partial evaluation are

- forward and backward data structure propagation.
- opening of procedure calls (unfolding).
- evaluation of builtins, whenever possible.

The improvements in the program resulting from partial evaluation are essentially due to the lower number of procedure calls and to specialization for the "input" partial values.

Logic languages are naturally handled by partial evaluation, since the basic mechanisms (reduction and forward-backward data structure propagation) are already in the standard interpreter (resolution and unification). Unification directly supports forward and backward data structure propagation and input values are not forced to be constant values but can be partially determined data structures (i.e., terms containing logical variables).

The first attempt to apply partial evaluation to logic based Knowledge Base Management Systems was in the area of data base query optimization /Venken84/. Partial evaluation techniques were recently applied to metaprograms in the framework of PROLOG /Takeuchi86a, Gallagher86/, Flat Concurrent Prolog /Takeuchi86b, Safra86/ and of a functional language /Jones85/. It has been used to effectively derive an efficient compiler-compiler /Jones85/ and to define the various virtual machines of LOGIX /Safra86/. Our interest is verifying the feasibility of combining partial

evaluation with metainterpreters, used to define language extensions or tools. The same approach is being pursued at ICOT /Takeuchi86a/.

In partial evaluation of metaprograms, the partial input values are procedure calls. The partial evaluation of the metaprogram M applied to a call of the procedure P generates a specialization of M, which can be viewed as a new version of P. The new definition of P is a completely new procedure P', embodying some of the features relevant to M. This allows to replace a metacall to P (by means of M) by a direct call to P', since the direct execution of P' is equivalent to the execution of P through the metainterpreter M.

If M is the pure metainterpreter (without new inference rules or extended features), P' is equivalent to P. If the metainterpreter M contains additional inference rules (and the corresponding additional features), partial evaluation compiles the new features in the procedure P. For example,

- If M is the explanation metainterpreter, the procedure P' is a version of P, providing the explanation feature, when executed by the standard interpreter.
- If M is a debugger metainterpreter, P' is the version of P "instrumented" to allow the debugging with the standard execution.
- If M is a "query-the-user" metainterpreter, P' is the version of P which queries the user when executed by the standard interpreter.
- If the "new" language contains structuring concepts, such as theories and relations on theories, supported by metainterpreters embodying the corresponding inference rules, the language can be compiled to the original unstructured language.

In summary, the key aspects which makes partial evaluation interesting in the case of metaprograms are metacall translation to direct calls and "compile-time" evaluation of static knowledge. Partial evaluation allows then to combine the flexibility of metaprogramming with efficiency.

The Epsilon prototype contains a partial evaluator for full PROLOG extended with theories, specifically designed to act as a compiler for metaprograms. The "compilation" of one metainterpreter (inference engine) layer can better be understood by making reference to Figure 2, which shows that the theory T_3 resulting from the partial evaluation of T_1 has the same class (M_2) of the class (M_1) of T_1.The partial evaluator will not be described here (a technical description can be found in /Ghelfo86, Levi87/), since we are mainly interested in discussing its impact on the performance of inference engines defined by metaprograms. Let us only mention the fact that it embodies special features to efficiently handle metacalls and that it uses theories to store intermediate results, thus providing a richer analysis capability and a consequent richer set of equivalence preserving optimizing transformations. For example, this mechanism makes the treatment of "cut" quite easy.

8. An example of inference engine "compilation"

Let us consider the theory **bibrules** (in Figure 10), whose inference engine (the theory **engine**) contains the metaprogram (query-metainterpreter) **scall** (in Figure 5).

A call to the partial evaluator of the form **compile(bibrules,scall,callrules)** creates in the theory **callrules** the result of the partial evaluation of the goals

```
scall(brother(X,Y),bibrules),
scall(grand_mother(X,Y),bibrules),
scall(parent(X,Y),bibrules),
scall(uncle(X,Y),bibrules).
```

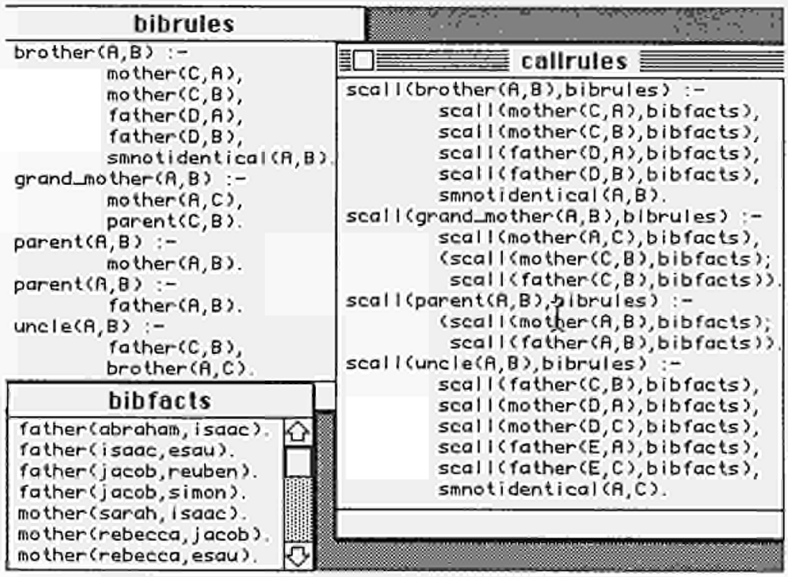


Figure 10. Two theories and the result of partial evaluation.

Links are also used to define which parts of the knowledge base are intended to be visible to partial evaluation. A program (theory) can then be made parametric with respect to some of the theories it uses or inherits.This is relevant to the optimization of incomplete knowledge, which can be encapsulated into a theory T*, which may be made visible to a theory T, by defining a suitable link interpreted by the inference engine of T. Such a link, however, is not visible to the partial evaluator, which, when applied to T, cannot evaluate all the references to T*. Thus the partial evaluation of T is independent from the current content of T*.

In our example, the theory **bibfacts** can be inherited from **bibrules** (link **clsinher**). If we make **bibfacts** not visible to partial evaluation, the result of **compile(bibrules, scall, callrules)** is that shown in the theory **callrules** in Figure 10. Programs in **callrules** are much more efficient than the application of the metainterpreter **scall** to the original programs in **bibrules**. For example, computing all the answers to the query **parent(X,Y)** in **bibrules** requires 138 logical inferences, while the same task in **callrules** requires 72 logical inferences only. All the metainterpreter components which can be "statically" evaluated disappear from the "compiled program". Note also that the metalevel simulation of the standard PROLOG interpreter (first 4 clauses of **proquery**) is not needed. In particular, since the main functor of the first argument of **scall** is known, all the accesses to clauses in **bibrules** can statically be solved and the only metacalls are those to the invisible theory **bibfacts**.

9. Another example: Forward chaining

The metaprogram contained in the inference engine **engine** in Figure 11 defines a procedure **forw**, which, given a new fact **Fact**, generates all its consequences in the theory **Th** and puts them in the theory **ThRes**.The procedure **forw** defines the following algorithm. For each clause in the theory (smthpreds(Th,P,N) is a backtrackable operation defined on a theory Th which returns the name P and the arity N of each predicate defined in Th), whose body contains an atom which is unifiable with **Fact**, if all the other atoms in the body can be solved in **Th**, asserts the instantiated

again the metainterpreter of Figure 5.

For example, let us consider the theories shown in the Figure 10. By evaluating **forw(father(isaac,jacob),bibrules,brut)** we obtain the theory **brut** in Figure 11.

The algorithm is very inefficient, because there exists no fast method to access the relevant clauses. All the considerations already made about the partial evaluation of the metainterpreter **scall** apply to **forw** too. In this case, however, the optimization on clause access is dramatic and allows to statically generate the "forward chaining" version of all the relevant clauses. As usual, the links between theories must explicitely be made visible to the partial evaluator. Let us consider the case where the link between **bibrules** and **bibfacts** is not visible.

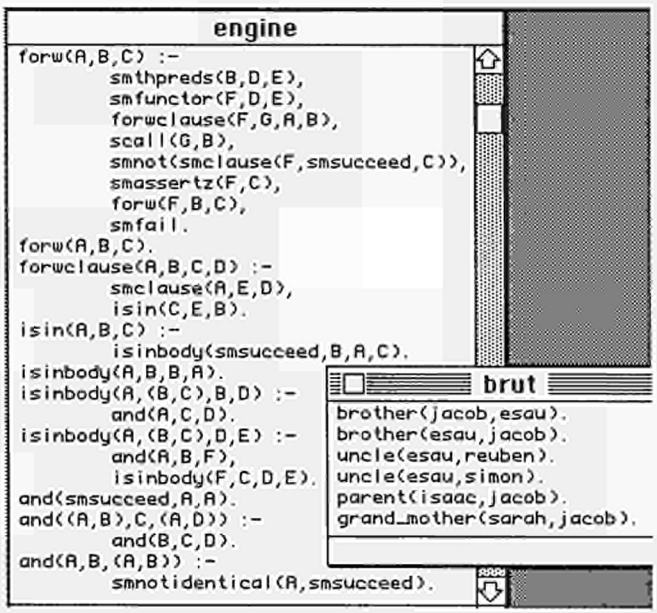


Figure 11. The metaprogram **forw** and the result of its application to **bibrules**.

The theory **tat** in Figure 12 contains the result of the partial evaluation of **forw(father(X,Y),bibrules,Z)**. Note that **forw** has been dramatically simplified and the only metacalls are those related to the invisible theory **bibfacts** . The result of the partial evaluation is therefore parametric with respect to the content of **bibfacts**, since we partially evaluate **forw** only with respect to the general knowledge contained in the theory **bibrules**.

If **bibfacts** is made visible to the partial evaluation of **forw(father(X,Y),bibrules,Z)**, we obtain the more efficient version (not containing any call to **scall**), contained in the theory **frulfacts** in Figure 13. **frulfacts** now depends both on **bibrules** and on **bibfacts**, and must be updated whenever one of the two theories is updated. Let us conclude with some experimental performance results. The query **forw(father(isaac,jacob),bibrules,brut)**, if executed in **engine**, requires 2567 logical inferences. The same query requires 459 logical inferences in **tat** (partial evaluation with respect to **bibrules** only) and 22 logical inferences only in **frulfacts** (where partial evaluation considers **bibfacts** too).

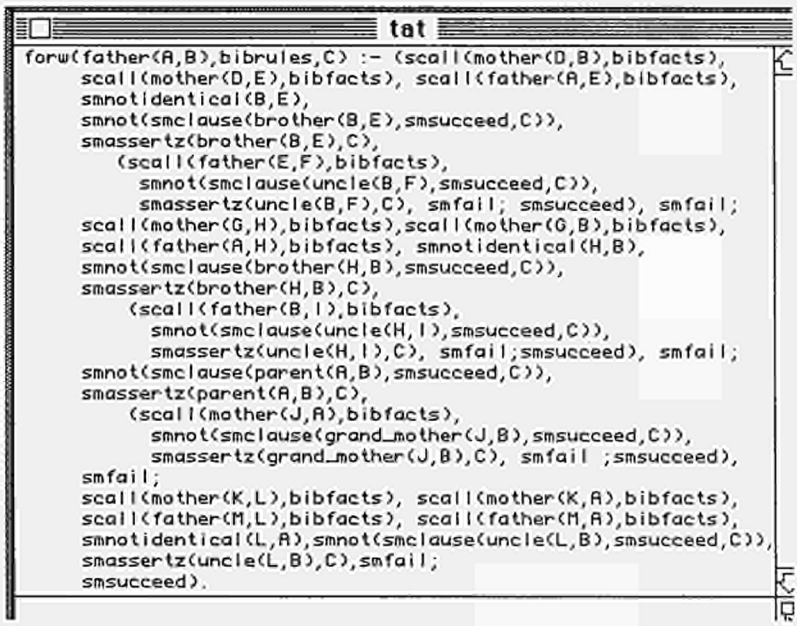


Figure 12. The result of the partial evaluation of **forw(father(X,Y),bibrules,Z)**.

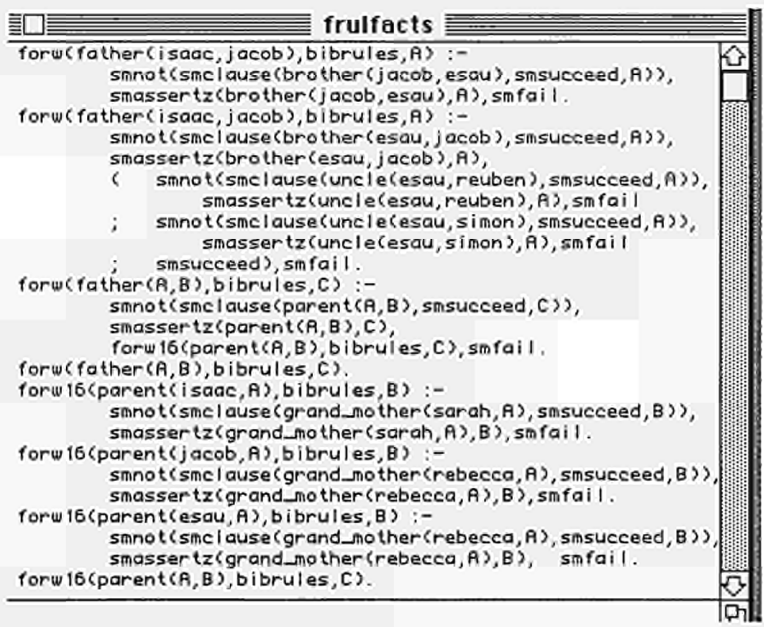


Figure 13. The partial evaluation of **forw(father(X,Y),bibrules,Z)** (**bibfacts** visible).

10. Theories and Data Bases

In connection with data bases, the theory feature is used to represent

- data base theories, which stand for sets of tuples stored in an external data base.
- data dictionaries, i.e. the metalevel descriptions of data base theories, which contain the knowledge about what is contained in data base theories. A data dictionary is linked to its data base theory by a specific link.
- integrity constraints.
- the results of queries to the data base inference engine. The result of a query to a data base is a logic program representation of the answers, stored in a suitable theory.

Let us first consider the naive one-tuple-at-a-time interpretive solution, in the case shown in Figure 4, where T_1 contains a logic program, T_2 is a data base theory, M_1 is the logic program inference engine and M_2 is the data base processor. Note that no metaknowledge on the data base is required (apart from verification purposes), since T_1 directly inherits from the data base. If the query processor in M_1 cannot solve a subquery in T_1, it activates the data base query processor on T_2, which will get all the relevant tuples from T_2 and return the answers in a logic theory T3. The query processor in M_1 will then try to solve the query in T_3.

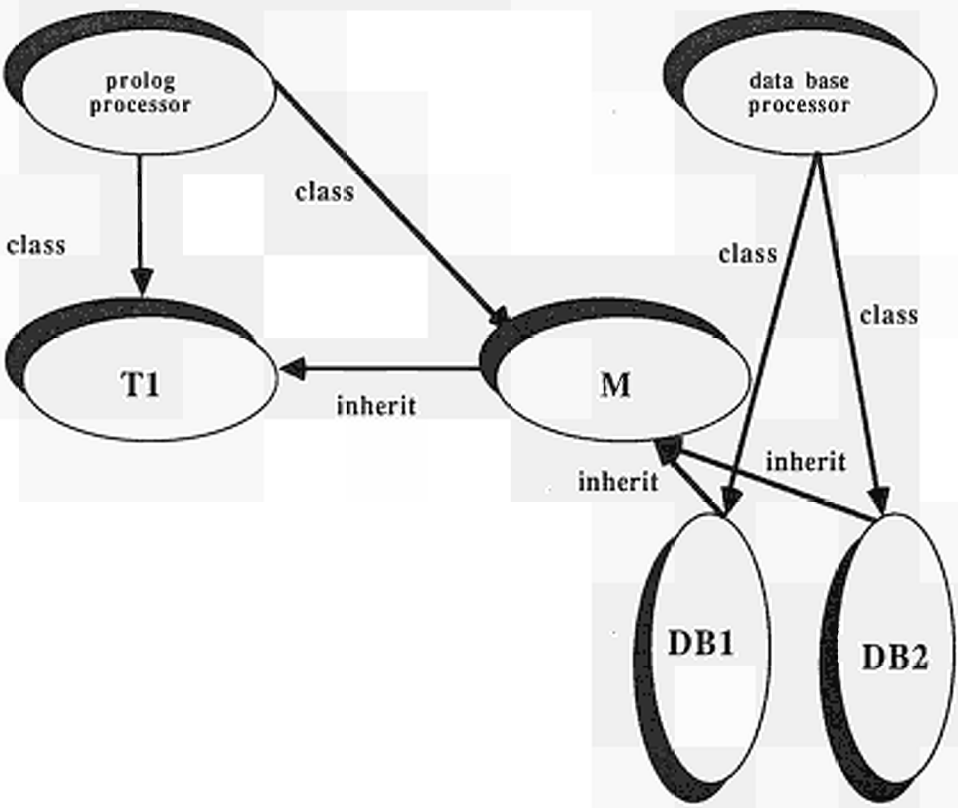


Figure 14. The uniform view of several data base theories.

In the general case, the knowledge base will contain several data base theories. In such a case,

it is certainly convenient to use the metaknowledge about the different data bases to efficiently find the relevant data base. This can be achieved by inheriting from the logic program a single theory M (the logic program view of all the data bases), which contains such a metaknowledge. As shown in Figure 14, M is linked to the data bases (and possibly to the corresponding data dictionaries).

The above solution allows to optimize the data base access, by grouping the queries to the same data base and by performing joins on different data bases within the communication (data base) processor.

11. Implementation of the communication between the kernel and data bases

The whole communication process is detailed in the following:

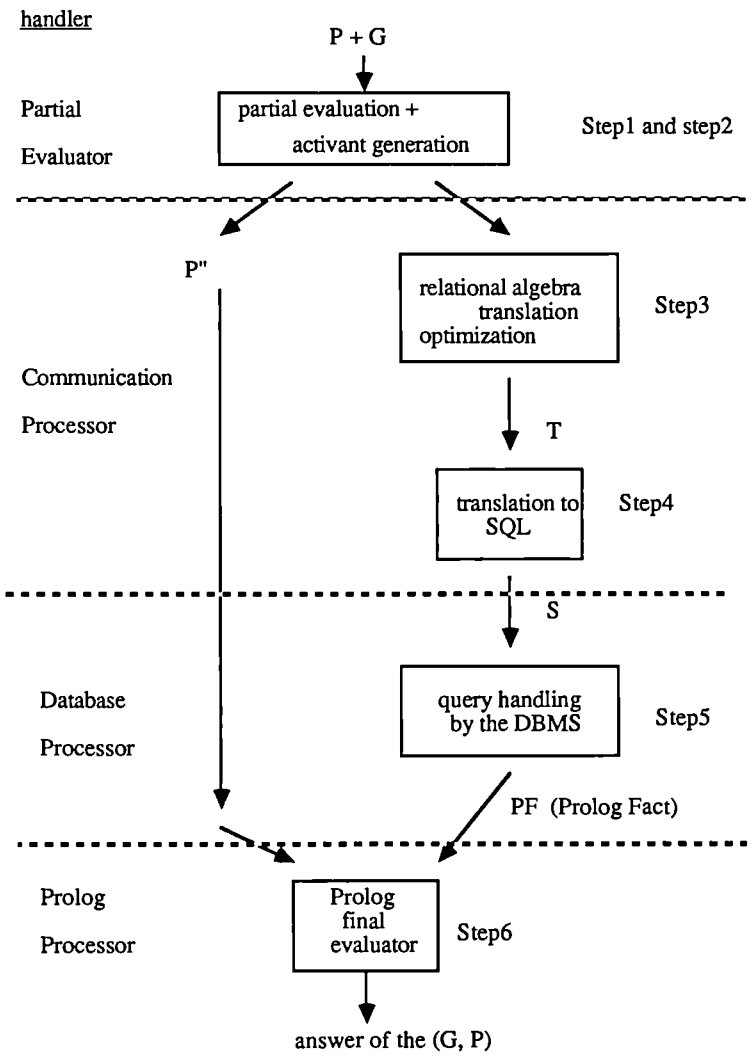


Figure 15

The communication betweeen the kernel machine and DBMSs in the EPSILON prototype is made through a piece of software called the communication processor (C.P.).

The major features of the C.P. are :

- The existence of specific language, which is mainly based on the **relational algebra**. The aim of this language is twofold :

  ◊ First of all, it insures the correct transfer of the semantics of the Processor-calls to dbcalls (see report P530-6). In that sense, it is only a translation medium between the kernel and DBMSs. The semantic overlap on the DBMS languages has to be as large as possible.

  ◊ But it is also a means to express data transformations to be made by the C.P. : for example, join operations on data coming from distinct data bases or treatment of closed queries, that is queries without variables which are to be handled in a special way by the C.P.

- The use of optimization techniques throughout the communication process since the drawback of the generality of the architecture is its lacks of efficiency if implemented without care. Optimizations are introduced at the processor-call generation level (use of partial evaluation techniques, of semantic constraints on data), at the dbcall generation level (reordering or simplification of the relational operator trees representing the queries).

In order to minimize the inference engine's calls to Databases, partial evaluation techniques have been used to gather elementary calls into global calls and to have a maximal instantiation of calls (VENKEN 84).

The communication is illustrated by the linear graph given in figure 15.

- The **STEP1** which is a **partial evaluator** uses the source program P and a goal G as input (P is a full Prolog program, with recursive predicates, builtins and functions).

  In our case, the partial evaluator will:

  ◊**1 collect and delay edb-predicates** to be sent to database system which is the same purpose as in compiled approaches. This point is very important to solve the first problem.

  ◊**2 instantiate the variables** in the edb-predicates as much possible, that is to solve the last problem.

  ◊**3 evaluate the builtins** whenever possible. That is, in the system, user is allowed to use a large number of primitives in his program.

  Evaluating builtins whenever possible is an important optimization issue in this technique.

  The result of the partial evaluation is still a program P' in logic, but the edb-predicates in user's program have been collected. The variables in the edb-predicates have been partially instantiated, and several primitives have been evaluated. That is, the source program has been optimized.

- The **STEP2** is the **activant generator** which uses the program P', that is the result of STEP1, as input. The activant generator separates the input program P' into two, the program P" where the conjunction of edb-predicates are changed to activants (a collection of dbcalls), and a set of activants.

- The **STEP3** is the **activant translation** which only uses the set of activants as input.

In this step, the set of activants is translated into a program written in an intermediate language, relational algebra language. Some optimizations work can be done in this step.

- In the **STEP4**, the program is then transformed into a program called PDB program written in a database system language (after reordering of the relational expression).

- The **STEP5** is a general **database system** which uses PDB program as input. Then a DB result translator will translate the result of DB system into a set of facts in Prolog form. Then, the set will be loaded to the work-space of Prolog.

- The **STEP6** is done by the **Prolog interpreter** which uses the facts loaded from the database system and the program P" generated by partial evaluation and activant generation (STEP1 and STEP2), as input. In a further stage, final evaluation could be meta-controlled to process synchronization between bankfeeding mechanisms and inference processing.

12. Generation of the activants

The task of the activants generator is to group the dbcalls and to rename the dbcall-groups by so called "activants" predicates. The activants are the predicates which refer to facts brought from the external data base to the inference engine work bases via the communication processor.

Let us consider the program obtained at the exit of the partial evaluator.

Program (Venken's example)

```
prg(_x,violette): F(_x,anna).
prg(_x,jan)  :   F(_x,violette).
prg(_x,stanis): F(_x,violette).
prg(_x,henry): F(_x,henriette).
prg(_x,_y)   :   F(_x,_z) & F(_z,_y).
prg(_x,_y)   :   F(_x,_y).
```

An activant is defined for each db-call group.

The result is:

```
prg(_x,violette) : a1(_x).
prg(_x,jan)      : a2(_x).
prg(_x,stanis)   : a2(_x).
prg(_x,henry)    : a3(_x).
prg(_x,_y)       : a4(_x,_y).
prg(_x,_y)       : a5(_x,_y).
```

With the definitions of activants:

```
a1(_x)   : F(_x,anna).
a2(_x)   : F(_x,violette).
a3(_x)   : F(_x,henriette).
a4(_x)   : F(_x,_z) & F(_z,_y).
a5(_x)   : F(_x,_y).
```

Note that this solution allows to process every selection operation by the database management system, which is much more efficient than PROLOG for this purpose. Facts returned from the Data Base systems will be put in various work bases, decreasing the work of inference engine at unification time.

13. Translation from Prolog to relational algebra

Once generated, activants have to be translated into the pivot language of the communication processor, that is into R-Algebra.

Several hypothesis are set upon the rules to be translated:

- predicates are relational or comparison predicates (= > < => =<).
- the function names are instantiated.
- predicates are function free, that is their parameters are constants or variables.
- there is no constant in the head of clause.
- variables of the head have a single occurrence in the head:
  (p(_x,_x) is forbidden).
- all the variables in the head have at least one occurrence in the queue.

The translation is then done in two steps:

First step is a "flat translation" using Ceri's method /CERI-85/.
Second step is a re-structuration of the relational-algebra tree, regrouping relations of the same database in the same subtrees, and including comparison predicates.

Example:

p(_x) : r(_x, _y) & <(_y, b) & s(_x, a) & t(_y, _z).
R and T are in database 1.
S is in database 2.
from step one: $P = \prod_1 \text{sigma}_{\text{ind(1)=ind(3),ind(2)=ind(5),ind(4)=val(a)}} R \times S \times T$
by adding <(_y, b):
$P = \prod_1 \text{sigma}_{\text{ind(1)=ind(3),ind(2)=ind(5),ind(4)=val(a),ind(2)<val(b)}} R \times S \times T$
by grouping relations of the same database:
$P = \prod_3 \text{sigma}_{\text{ind(3)=ind(5),ind(4)=ind(5),ind(1)=val(a),ind(4)<val(b)}} (T \times R) \times S$
note that the index have changed because of the reordering.
by generating joins:
$P = \prod_3 \text{sigma}_{\text{ind(6)=val(a),ind(1)<val(b)}} \text{join}([3=1], \text{join}([1<b], T, R), S)$
note once more the re-ordered index.

In conclusion, the Relational expression is ready to be optimized and then translated or even directly translated.

14. Database query generation

One specific task of the C.P. is the translation of relational algebra expressions (which are equivalent to the set of activants) to database manipulation languages, in our case SQL.
Translating relational algebra expression to SQL queries is rather easy.

15. Conclusions

In the near future, the Epsilon prototype will include other features that are currently investigated by all the partners in the Epsilon Project (Bense KG, C.R.I.S.S., University of Dortmund, University C. Bernard of Lyon, University of Pisa and Systems & Management). These include more inference engines and tools, a general-purpose integrity constraint verification mechanism and a more powerful graphical interface. Two more important related activities are the experimental application of the Epsilon approach to some test problems and the design of a distributed version of the current prototype.

16. References

/Ceri-85/: **S. Ceri, G. Gottlob, L. Lavazza** "Transformation and optimization of logic queries: the algebraïc approach".Internal report- Politecnico di Milano.

/Coscia86/ **P. Coscia, S. Djennaoui, P. Franceschi, J. Kouloumdjian, G. Levi, L. Lei, G.-H. Moll, I. de Saint Victor, G. Sardu, C. Simonelli and L. Torre,** The Epsilon Knowledge Base Management System: Architecture and Data Base access optimization, Workshop on Integration of Logic Programming and Data Bases (Venezia, December 1986).

/Coscia87/ **P. Coscia, P. Franceschi, G. Levi, G.Sardu and L. Torre,** Object level reflection of inference rules by partial evaluation, to appear in *Meta-level architectures and reflection*, D. Nardi and P. Maes, eds. (North-Holland 1987).

/Furukawa84/ **K. Furukawa, A. Takeuchi, S. Kunifuji, H. Yasukawa, M. Ohki and K. Ueda,** Mandala: A logic based knowledge programming system, Proc. Int'l Conf. on Fifth Generation Computer Systems (1984), 613-622.

/Gallagher86/ **J. Gallagher,** Transforming logic programs by specializing interpreters, Proc. ECAI86.

/Gardarin-84/: **G. Gardarin** "Bases de données : les systèmes et leurs langages". Ed. EYROLLES 1984.

/Ghelfo86/ **S. Ghelfo and G. Levi,** A partial evaluator for metaprograms in a multiple theories logic language. Epsilon Project Report (October 1986).

/Jones85/ **N.D. Jones, P. Sestoft and H. Sondergaard,** An experiment in partial evaluation: The generation of a compiler-compiler. First Int'l Conf. on Rewriting Techniques and Applications, LNCS 202 (1985), 124-140.

/Kauffman86a/ **H. Kauffmann and A. Grumbach,** Representing and manipulating knowledge within "worlds". Proc. First Int'l Conf. on Expert Data Base Systems, L. Kershberg, Ed. (1986), 61-73.

/Kauffman86b/ **H. Kauffmann and A. Grumbach,** MULTILOG: MULTIple worlds in LOGic programming, Report (1986).

/Levi87/ **G. Levi and G. Sardu,** Partial evaluation of metaprograms in a "multiple worlds" logic language, submitted to the Workshop on Partial and Mixed Computation (Denmark, October 1987).

/LIJK-86/: **Li Lei, J. Kouloudjian** "An implementation of a partial evaluation system". Internal report -University of Lyon (1986).

/Safra86/ **S. Safra and E. Shapiro,** Meta-interpreters for real, Information Processing-86, H.-J. Kugler, Ed. (Elsevier Science Publishers B.V., 1986).

/Smith-75/: **J. M. Smith, P.V. Chang** "Optimizating the performence of a relational algebra database interface". Con. A.C.M. V18 ,N°10, pp 568-579.

/Sterling84/ **L. Sterling,** Expert system = Knowledge + Meta-interpreter, Tech. Rep.CS84-17, Weizmann Institute of Science, Rehovot, Israel (1984).

/Takeuchi86a/ **A. Takeuchi and K. Furukawa,** Partial evaluation of PROLOG programs and its application to metaprogramming, Information Processing-86, H.-J. Kugler, Ed. (Elsevier Science Publishers B.V., 1986), 415-420.

/Takeuchi86b/ **A. Takeuchi,** Affinity between meta interpreters and partial evaluation, Information Processing-86, H.-J. Kugler, Ed. (Elsevier Science Publishers B.V., 1986).

/Ullman-80/: **J.D. Ullmann** "Principles of database systems". Computer science press -1980.

/Venken84/ **R. Venken,** A PROLOG meta-interpreter for partial evaluation and its application to source-to-source transformation and query optimization. Proc. ECAI-84: Advances in Artificial Intelligence (T. O'Shea, Ed.), North-Holland 1984, 91-100.

Project No. 1106

INTRODUCTION TO PROLOG III

Alain COLMERAUER

Groupe Intelligence Artificielle, Unité de recherche Associée au CNRS 816, Faculté des Sciences de Luminy, 70 route Léon Lachamp, Case 901, 13288 Marseille Cedex 9, France.

Abstract. The Prolog III programming language extends Prolog by redefining the fundamental process at its heart; unification. Prolog III integrates into this mechanism, refined processing of trees and lists, number processing, and processing of complete propositional calculus. We present the specifications and the logico-mathematical model for this new langauge, in which we replace the notion of unification by the more appropriate concept of constraint resolution. The capabilities thus acquired by the language are illustrated by various examples.

INTRODUCTION

Prolog was initially designed to process natural languages [3]. Its use to solve problems in increasingly varied areas has brought out its qualities, but have also made clear its limits. Some of these limitations have been by-passed using more and more efficient implementations and ever richer environments. The fact remains that the core of Prolog, Alan Robinson's unification algorithm [8], has not changed over 15 years, and is becoming less and less significant compared with an ever-increasing number of external procedures. The best examples of such procedures are number processing. These external procedures are unfortunately difficult to use. To call them, one must be sure that certain parameters are known, and this clashes with the general Prolog philosophy in which it is possible anywhere and at any time to talk about an unknown object x.

I therefore decided to fundamentally reshape Prolog by integrating the following features at the unification level: (1) refined manipulation of trees, including infinite ones, and lists processing, (2) complete processing of Boolean algebra, ie propositional calculus, (3) numerical processing including addition, substraction, multiplication by a constant and the relations $<$, $\leq$, $>$, $\geq$, (4) general processing of the relation $\neq$. As was the case when we described Prolog II [4], which already integrated infinite trees and the relation $\neq$, this reshaping consists of the replacement of the unification concept by the concept of constraints resolution in a specific domain equipped with precise operations and relations.

The aim of this article is therefore to describe the foundations of a new language named Prolog III, and to illustrate its capabilities with some examples. Of course Prolog III is more than an

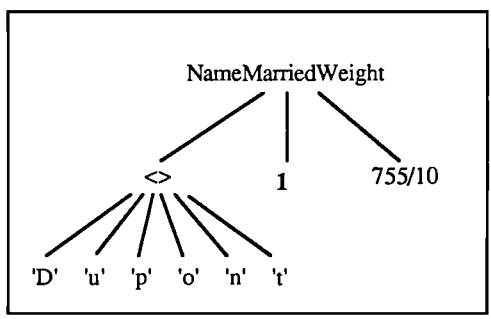
exercise in thought. A prototype for its interpreter has already begun to run and has been used to test the examples given here. This prototype was produced jointly by GIA - our laboratory -, and the company PrologIA. Important support was provided both by the National Research Center of Telecommunications (contract 86 1B 027) and by the European Community, within the framework of an ESPRIT project (P1219, 1106) in which the other partners are Mercedes, Bosch and GIT.

THE POSSIBLE VALUES OF A VARIABLE

In standard programming languages, each variable is given a value. When a program is run, these values are modified successively by means of assignment instructions, and final values are obtained for the different variables. In Prolog III, a variable represents an unknown value, and when a program is run, its aim is not to modify this value, but to determine it. So a variable behaves like an unknown quantity in a mathematical equation, like x for example in $x = (1/2)x+1$. The main difference is that x will represent something rather more complex than a number: a *tree*. These trees will be made up of nodes labelled by:

(1) identifiers,
(2) the double sign <>,
(3) boolean values,
(4) rational numbers,
(5) characters.

Here is one



The boolean values are denoted by **1** and **0** and rational numbers are represented by fractions. The number of branches emanating from each tree node is finite, and these branches are ordered from left to right. However a branch can be infinite. A tree consisting of only one node is a *leaf*,

and no difference will be made between a leaf and the label it carries. Therefore, boolean values and rational numbers will be considered as special types of trees. A tree whose initial node is the sign <> is a *list* and is used to represent the finite sequences of trees which constitutes its immediate sons. Thus the leaf labelled <> represents the empty list.

KNOWN CONSTANTS

To represent our trees we will have at our disposal (1) variables, which will always be written in italics to distinguish them from identifiers, (2) constants, used to name certain types of trees, and (3) operations for constructing trees from other trees. A tree will therefore be represented by a formula which includes these three elements: variables, constants and operations. This formula, considered as a syntactical object, will be called a *term*. The known constants are:

(1) identifiers such as,

Peter, LightMeal, calculus12,

(2) the empty list,

<>

(3) the boolean values

0, 1,

(4) positive or zero integers, such as,

0, 1,2 ,1987,

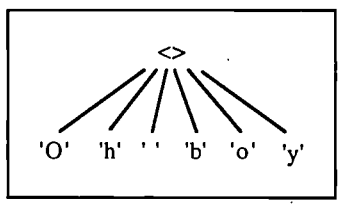
(5) characters such as,

'a', B', '4', '<',

(6) non-empty strings, such as the string

"Oh boy"

which is the tree



It should be noted that the only numerical constants are positive or zero integers. Other rational numbers will be represented using positive integers, the operation of division and the operation of changes of sign which we will introduce later. It is pointless to introduce a specific constant for the empty string, since such a string is the same as the empty list <>.

KNOWN OPERATIONS

We can now proceed to enumerate the known operations. An operation will simply be a mapping from a subset of tree tuples into the set of trees. We will suppose that all the tuples in this subset are of the same length, this length being the arity of the operation. An operation of arity n can be schematized by a formula having the form

$$a_1...a_n \rightarrow f(a_1...a_n),$$

where $f(a_1...a_n)$ designates the notation used to represent the tree which results from applying this operation on the tuple $a_1...a_n$. We have three types of operation at our disposal: boolean operations, arithmetical operations, and operations to construct complex trees.

Boolean operations are only defined if the operands are boolean values, in other words, leaves labelled by boolean values. There are four of them:

(1) the *not*,

$$a_1 \rightarrow \neg a_1,$$

(2) the *and*,

$$a_1a_2 \rightarrow a_1 \wedge a_2,$$

(3) the (non exclusive) *or*,

$$a_1a_2 \rightarrow a_1 \vee a_2,$$

(4) the operation which produces **1** if the two operands are equal and **0** if they are not,

$$a_1a_2 \rightarrow a_1 \equiv a_2.$$

The arithmetical operations are only defined if the operands are rational numbers, in other words leaves labelled by fractions (or integers), and in the case of division, only if the second operand is not 0. There are six of them:

(1) the neutral operation,

$$a_1 \rightarrow +a_1,$$

(2) change of sign,

$$a_1 \rightarrow -a_1,$$

(3) addition,

$$a_1a_2 \rightarrow a_1+a_2,$$

(4) subtraction,

$$a_1a_2 \rightarrow a_1-a_2,$$

(5) multiplication,

$$a_1a_2 \rightarrow a_1 \times a_2,$$

(6) division,

$$a_1a_2 \rightarrow a_1/a_2.$$

There will be a *linearity* restriction on the terms involving these operations: in a multiplication

only one of the two operands can contain variables, and in a division, the second operand must not contain variables. For the sake of convenience, we write a_1a_2 instead of $a_1 \times a_2$ as long as this creates no confusion.

The construction operations make it possible to construct trees which are not necessarily reduced to a leaf. There are four types:

(1) list constructions, for all values n such as $n \geq 1$,

$$a_1 \ldots a_n \rightarrow \langle a_1, \ldots, a_n \rangle,$$

(2) tree constructions, for all values n such as $n \geq 2$,

$$a_1 a_2 \ldots a_n \rightarrow a_1(a_2, \ldots, a_n),$$

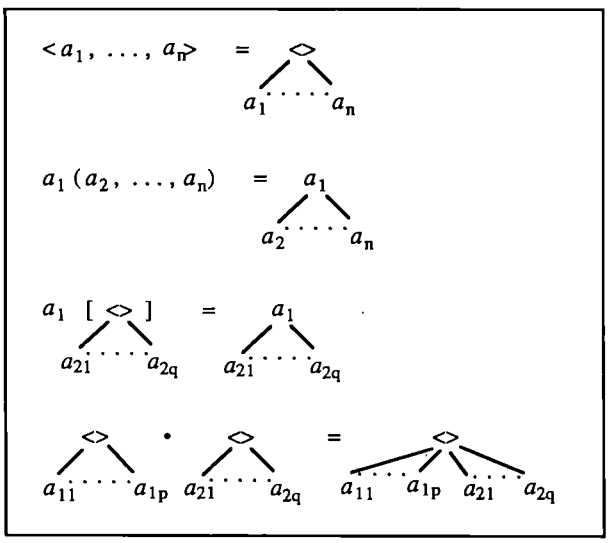
(3) general tree construction,

$$a_1 a_2 \rightarrow a_1[a_2],$$

(4) list concatenation,

$$a_1 a_2 \rightarrow a_1 \cdot a_2.$$

List constructions are defined whatever the trees a_i are. Tree constructions are only defined if the tree a_1 is a leaf. General construction of a tree is only defined if tree a_1 is a leaf and tree a_2 is a list. Concatenation is only defined if the two trees a_1 and a_2 are lists.The exact functioning of all these operations is summarized in the diagram below:



The following equalities can be noted:

$$\langle a_1, \ldots, a_n \rangle = \langle\rangle(a_1, \ldots, a_n),$$
$$a_1(a_2, \ldots, a_n) = a_1[\langle a_2, \ldots, a_n \rangle]$$

and also, when a is a list,

$$<>[a] = a,$$

$$<> \cdot a = a \cdot <> = a.$$

It can also be seen that in order to represent a list whose first element is e, and the remainder of which is x, we write

$$<e> \cdot x$$

and that

$$e[x]$$

designates absolutely any tree, where the label for its initial node is e and the list of its sons, which can be empty, is x.

As with division and multiplication, we place an important constraint on list concatenation: in a concatenation, if the operand on the left is a variable x, then the length n of the list x must be known and explicitly specified by a constraint of the form $x{:}n$ which we will introduce in the next section. In regard to this restriction, it should be borne in mind that concatenation is an associative operation, and that here we make no difference between $(x \cdot y) \cdot z$ and $x \cdot (y \cdot z)$.

With the help of the constants and the operations we have introduced, we can represent our first example of a tree equally well using either of the following two terms:

NameMarriedWeight(<'D','u','p','o','n','t'>, **1**, 755/10),

NameMarriedWeight("Dupont", **1**, 75+5/10).

KNOWN RELATIONS

A certain number of binary and unary relations are known. Binary relations enable you to express the following constraints on trees:

$a_1 = a_2$, trees a_1 and a_2 are equal,

$a_1 \neq a_2$, trees a_1 and a_2 are different,

a_1 => a_2, trees a_1 and a_2 are booleans and if a_1 equals **1** then a_2 equals **1**,

$a_1 < a_2$, trees a_1 and a_2 are numbers and a_1 is strictly less than a_2,

$a_1 > a_2$, trees a_1 and a_2 are numbers and a_1 is strictly greater than a_2,

$a_1 \leq a_2$, trees a_1 and a_2 are numbers and a_1 is less than or equal to a_2,

$a_1 \geq a_2$, trees a_1 and a_2 are numbers and a_1 is greater than or equal to a_2.

The terms "booleans" and "numbers" that we use here refer to leaves labeled by boolean values and leaves labeled by rational numbers.

Unary relations enable you to express the following constraints:

a : **fact**, the initial node of tree a is labelled by an identifier,

a : **list**, tree a is a list,

a : n, tree a is a list of known length n,

a : **string**, tree a is a string,

a : **leaf**, tree a is a leaf,

a : **id**, tree a is a leaf labelled by an identifier,

a : **bool**, tree a is a leaf labelled by a boolean,

a : **num**, tree a is a leaf labelled by a rational number,

a : **char**, tree a is a leaf labelled by a character.

It should be noted that we did not really need the relations **list**, **leaf**, **bool** and **num**, since if we want to constrain x to represent a list, a leaf, a boolean leaf or a numerical leaf, this can be done by simply replacing one of its occurrencies by the following terms:

$$<>\cdot x\,,\quad x[<>],\quad \neg\neg x\,,\quad +x.$$

This is because the operations we use here include restrictions on the nature of their operands and thus create implicit typing of the variable x.

SYSTEMS OF CONSTRAINTS

Constraints are used to construct *systems* of constraints, that is finite sets of constraints which must all be satisified at the same time. The first thing Prolog III enables you to do is solve these systems. For example, to find out the number x of pigeons and the number y of rabbits required to have a total of 12 heads and 34 legs, all you need to do is write the query

$$\{x \geq 0, y \geq 0, x+y = 12, 2x+4y = 34\}?$$

and the machine will answer

$$\{x = 7, y = 5\}.$$

To compute a list z of 10 elements that will produce the same result no matter wether <1,2,3> is concatenated to its the left or <2,3,1> is concatenated to its right, you merely need to write the query

$$\{\ z\colon 10,\ \ <1,2,3>\cdot z = z\cdot<2,3,1>\}?$$

The answer is

$$\{z = <1,2,3,1,2,3,1,2,3,1>\}$$

We can also solve both problems at the same time, by asking

$$\{x \geq 0, y \geq 0, z\colon 10, \text{trio}(x+y,2x+4y,<1,2,3>\cdot z\ ) = \text{trio}(12,34,\ z\cdot<2,3,1>)\}?$$

where "trio" is any identifier. The answer is

$$\{x = 7, y = 5, z = <1,2,3,1,2,3,1,2,3,1>\}.$$

So the heart of a Prolog III interpreter will consist of a general algorithm for the resolution of system of constraints. This algorithm will be used to decide whether a system is solvable, that is, whether it is possible to attribute values to its variables so that all the constraints are satisfied. If the system is solvable, this algorithm will be also used to simplify it, so that its solutions, that is the values of its variables become apparent. These values may be unique, as in the previous examples, but can also be multiple, as they are in the following three simplified systems:

$$\{0 \leq x, x \geq 3/4, x \neq 1/2\},\ \{y => z\},\ \{u = \text{father}(v)\}.$$

If there are no limits on these values, the simplified system will be the empty system denoted by { }. The constraints resolution algorithm replaces the unification algorithm used in a standard Prolog. Because of this, it must be very efficient and so reliable that it becomes a black box for the programmer. These are the two factors which dictated our choice of known operations, known relations and our restrictions on multiplication, division and concatenation.

On this subject, we can note that in Prolog III a constraint does not represent a boolean value. It may or may not be satisfied by certain values of variables. Therefore it is not possible to write $(x<2)\vee(x>1)$. Moreover, our signs $\neg$, $\wedge$,$\vee$, $\equiv$ are neither connectors nor relations, but operations. To write $(x<2)\wedge(x>1)$, we will write $\{x<2,x>1\}$, and to assert $x\vee y$ we will write $\{x\vee y=1\}$. However we can write $\{x=>y\}$ since => has been defined as a relation. We can also note that in Prolog III there is no unary relation which makes it possible to constrain a rational number to be an integer.

THE MEANING OF A PROGRAM FOR THE PROGRAMMER

We can now explain in general terms what a Prolog III program is. Basically it consists of a recursive definition of a subset of trees. Each element in this subset is called a *fact*, and represents a proposition which is considered true by the programmer. An example of such a proposition could be "Dupont is married and weighs 75,5 kg", which could be represented by our first example of a tree. The set of facts defined by a program is usually infinite and in a way constitutes an enormous hidden database. The execution of a program aims to reveal certain parts of this database.

Strictly speaking, the program is a set of *rules*: the facts that the programmer wants to define obey these rules. Each rule has the form

$$t_0 \rightarrow t_1 \ldots t_n\ ,\ S$$

where n can be zero, where the t_i's are terms, and where S is a system of constraints which may be absent (in which case it is considerd to be the empty system). Here is such a set of rules; this is our first example of a Prolog III program:

LightMeal(a, m, d) →
    Appetizer(a, i) Main(m, j) Dessert(d, k),
    {$i \geq 0, j \geq 0, k \geq 0, i+j+k \leq 10$};

Main(m, i) → Meat(m, i);
Main(m, i) → Fish(m, i);

Appetizer(radishes, 1) →;
Appetizer(pâté, 6) →;

Meat(beef, 5) →;
Meat(pork, 7) →;

Fish(sole, 2) →;
Fish(tuna, 4) →;

Dessert(fruit, 2) →;
Dessert(icecream, 6) →.

It is an improvement on a program which is perhaps too well-known [4], but which remains an efficient pedagogical tool: the calculation of the components of a meal having a caloric content below a certain amount. Note the fact that each food type has a specific caloric value.

The variables which appear in the rules are of course quantified universally, in other words each rule $t_0 \rightarrow t_1 \ldots t_n, S$ is simply an abbreviated way of writing all the *instantiated* rules

$$a_0 \Rightarrow a_1 \ldots a_n$$

obtained by giving the variables all possible values which satisfy system S and which transform the terms t_i into well-defined trees a_i. Here are a few fragments of the instantiated rules for the above program:

.....
LightMeal(pâté,sole,fruit) ⇒
    Appetizer(pâté,6) Main(sole,2) Dessert(fruit,2) ;
.....
.....
Main(sole,2) ⇒ Fish(sole,2) ;
.....
.....
Appetizer(pâté,6) ⇒ ;
.....

.....
Fish(sole,2) ⇒ ;
.....
.....
Dessert(fruit,2) ⇒ ;
.....

Naturally, the process of *instantiating* rules is purely a mental process on the part of the programmer. The machine will not work in this way, on the contrary, when it uses the rules it will try to keep them as general as possible. However, the instantiated rules do not make use of variables or constraints, but only trees. Their meaning is therefore much clearer.

Each instantiated rule $a_0 \Rightarrow a_1 ... a_n$ can be interpreted in two ways:
(1) as a *rewrite rule*: any occurrence of the tree a_0 in a sequence of trees can be replaced by the sequence of trees $a_1 ... a_n$ (when $n = 0$ this has the same effect as the deletion of a_0 from the sequence);
(2) as a *logical property* of a subset E of trees: if all the trees a_1 and ... and a_n belong to the subset E then the tree a_0 also belongs to E (when $n = 0$ this property is reduced to: the tree a_0 belongs to E).

As we have seen, a program represents the accumulated set of instantiated rules which originate from all its rules. Depending on which of the two above interpretations we use, the *facts* defined by this program
(1) are the trees which can be deleted by a finite number of rewritings.
(2) form the smallest subset of trees (in the sense of inclusion) which satisfies all the logical properties.

When it is shown that these two definitions are equivalent it is natural to hesitate as to which direction the arrow should take in a rule. We prefer the arrow to go from left to right since this corresponds more closely to the method used by the machine. The fragments of the set of instantiated rules from the previous program make it possible to delete the tree

LightMeal(pâté,sole,fruit)

successively by

LightMeal(pâté,sole,fruit) ⇒
Appetizer(pâté,6) Main(sole,2) Dessert(fruit,2) ⇒
Main(sole,2) Dessert(fruit,2) ⇒
Fish(sole,2) Dessert(fruit,2) ⇒
Dessert(fruit,2) ⇒.

This tree is therefore a fact defined by the program. If we now treat these fragments of

instantiated rules as logical properties, we conclude successively that the three sets below are made up of facts defined by the program.

{Appetizer(pâté,6), Fish(sole,2), Dessert(fruit,2)},
{Main(sole,2)},
{LightMeal(pâté,sole,fruit)}.

MEANING OF A PROGRAM FOR THE MACHINE

We have now described the implicit information that is contained in a Prolog III program, but we have not yet explained how such a program is executed. The aim of the program's execution is to solve the following problem: given a sequence of terms $t_1...t_n$ and a system S of constraints, find the values of the variables which transform all the terms t_i into facts defined by the program, while satisfying all the constraints of S. This problem is submitted to the machine by writing the *query*

$$t_1...t_n, S\ ?$$

Two special cases are of particular interest. (1) If the sequence $t_1...t_n$ is empty then the query reduces itself to a request to solve the system S. We have already given some examples of such queries. (2) If the system S is empty (or absent) and the sequence of terms consists of only one term, the request can be restated as: what are the values of the variables which transform this term into a fact defined by the program. So if we now use the preceding example program, the query

LightMeal(a, m, d)?

will enable us to obtain all the sets of values for a, m, and d which consitute a light meal. In this case, the replies will be the following simplified systems:

{a=radishes, m=beef, d=fruit},
{a=radishes, m=pork, d=fruit},
{a=radishes, m=sole, d=fruit},
{a=radishes, m=sole, d=icecream},
{a=radishes, m=tuna, d=fruit},
{a=pâté, m=sole, d=fruit}.

We can explain the method used to calculate the replies to a given query by using an abstract machine. This is a nondeterministic machine whose function is described by these three forumlæ:

(1) $(W,\ t_0\ t_1...t_n\ ,\ S)$,

(2) $s_0 \rightarrow s_1.... s_m\ ,\ R$

(3) $(W,\ s_1...s_m\ t_1...t_n\ ,\ S \cup R \cup \{s_0=t_0\})$.

Formula (1) represents the state of the machine at any given moment. W is a set of variables whose values we want to establish, $t_1...t_n$ is a sequence of terms which we are trying to delete and S is a system of constraints which has to be satisfied. Formula (2) represents the rule in the

program that we are going to use to change the state of the machine. If necessary we rename some variables of formula (2), so that none of them occur in formula (1). Formula (3) is the new state of the machine after rule (2) has been applied. It is possible to progress to this new state only if the system of constraints in formula (3) has at least one solution in which each term of formula (3) is a well defined tree.

To describe the actual functioning of the Prolog III machine, we can say that it starts from an initial state (W, $t_0...t_n$,S), where W is the set of variables which appear in the query $t_0...t_n$,S and calculates all the states that can be arrived at by repeating the above process. Each time we arrive at a state where the sequence of terms is empty, we simplify the system of constraints with which it is associated and provide this as an answer. This final simplification can also be carried out on all the constraints systems as they are produced.

Let us now reconsider our first example program, and apply this process to the query

LightMeal(a,m,d)?

The initial state of the machine is

({a,m,d}, LightMeal(a,m,d), {}).

By applying the rule

LightMeal(a', m', d') → Appetizer(a', i) Main(m', j) Dessert(d', k),
{$i \geq 0, j \geq 0, k \geq 0, i+j+k \leq 10$}

we progress to the state

({a,m,d}, Appetizer(a',i) Main(m',j) Dessert(d',k),
{i≥0, j≥0, k≥0, $i+j+k$≤10, LightMeal(a,m,d)=LightMeal(a',m',d')})

which simplifies to

({a,m,d}, Appetizer(a',i) Main(m',j) Dessert(d',k),
{i≥0, j≥0, k≥0, $i+j+k$≤10, $a=a'$, $m=m'$, $d=d'$}),

then to

({a,m,d}, Appetizer(a,i) Main(m,j) Dessert(d,k), {i≥0, j≥0, k≥0, $i+j+k$≤10}).

By applying the rule

Appetizer(pâté, 6) →

and simplifying the result, we progress to the state

({a,m,d}, Main(m,j) Dessert(d,k), {a=pâté, j≥0, k≥0, $j+k$≤4}).

By applying the rule

Main(m', i) → Fish(m', i)

with a little simplification, we progress to

({a,m,d}, Fish(m',i) Dessert(d,k), {a=pâté, j≥0, k≥0, $j+k$≤4, $m=m'$, $j=i$}),

which then simplifies again to

({a,m,d}, Fish(m,j) Dessert(d,k), {a=pâté, j≥0, k≥0, $j+k$≤4}).

By applying the rule

Fish(sole, 2) →

we obtain

({*a,m,d*}, Dessert(*d,k*), { *a*=pâté, *m*=sole, k≥0, k≤2})

Finally, by applying the rule

Dessert(fruit, 2) →

we obtain

({*a,m,d*}, , {*a*=pâté, *m*=sole, *d*=fruit}).

We can conclude that the system

{*a*=pâté, *m*=sole, *d*=fruit}

constitutes one of the replies to the query.

To obtain the other replies, we proceed in the same way, but using the other rules. I should point out that there are a thousand ways of simplifying constraints and checking whether they are solvable. So it should not be assumed that the machine, which uses very general algorithms, makes the same simplifications as those that are shown above. But this is completely invisible for the programmer and thus of no importance.

Now we can go on to illustrate what Prolog III is capable of, with the help of other examples.

BANKING CALCULATION

In this example, the task set is the calculation of a series of successive instalments which have to be made to repay capital borrowed from a bank. We will assume that the same time period elapses between two instalments, and that during this period the interest imposed by the bank is 10%. The set of facts defined by the program will be the set of trees of the form

InstalmentsCapital(*x,c*)

where *x* is the list of instalments necessary to repay capital *c* with an interest rate of 10% between two instalments. The program itself can be summarized by two rules

InstalmentsCapital(<>, 0) →;

InstalmentsCapital(<*i*>•*x*, *c*) → InstalmentsCapital(*x*, (1+10/100)*c*-*i*);

The first rule expresses the fact that it is not necessary to pay instalments to repay zero capital. The second rule expresses the fact that the sequence of $n+1$ instalments to repay capital c consists of an instalment i and a sequence x of n instalments to repay capital c increased by 10% interest, but the whole reduced by instalment i.

This program can be used in different ways. One of the most surprising is to ask what value of i is required to have the sequence of instalments $<i,2i,3i>$ repay 1000$. All you need to do is write the query

InstalmentsCapital($<i, 2i, 3i>$, 1000)?

to obtain the reply

$\{i = 207 + 413/641\}$.

Here is an abbreviated trace of this calculation. Starting from the initial state

$(\{i\}$, InstalmentsCapital($<i,2i,3i>$,1000), $\{\})$,

we apply the rule

InstalmentsCapital($<i'>\bullet x$, c) $\rightarrow$ InstalmentsCapital(x, $(1+10/100)c-i'$)

and progress to the state

$(\{i\}$, InstalmentsCapital(x,$(1+10/100)c-i'$),

{InstalmentsCapital($<i,2i,3i>$,1000) = InstalmentsCapital($<i'>\bullet x,c$)}),

which simplifies to

$(\{i\}$, InstalmentsCapital(x,$(11/10)c-i'$), $\{i'=i, x=<2i,3i>, c=1000\})$,

then to

$(\{i\}$, InstalmentsCapital($<2i,3i>$,$1100-i$), $\{\})$.

The reader can verify that when the same rule is applied twice on this state, we obtain successively the states

$(\{i\}$, InstalmentsCapital($<3i>$,$1210-(31/10)i$), $\{\})$,

$(\{i\}$, InstalmentsCapital($<>$,$1331-(641/100)i$), $\{\})$.

By applying the following rule to the last state

InstalmentsCapital($<>$,0) $\rightarrow$

We finally obtain

$(\{i\}$, , $\{1331-(641/100)i = 0\})$

which simplifies to

$(\{i\}$, , $\{i=207+413/641\})$.

THE ART OF REASONING

The second problem makes use of Boolean algebra, and was provided by George Boole [2] himself. The aim is to show that "something has always existed" using the following 5 premisses:

(1) Something is.

(2) If something is, either something always was, or the things that now are have risen out of nothing.

(3) If something is, either it exists in the necessity of its own nature, or it exists by the will of another being.

(4) If it exists by the will of its own nature, something always was.
(5) If it exists by the the will of another being, then the hypothesis, that the things which now are have risen out of nothing, is false.

The set of facts defined by the program will be the set of trees having the form:

ValueOfSomethingHasAlwaysExisted(b)

where x designates a possible truth value for the proposition "something has always existed". We now introduce 5 boolean variables which are the possible truth values of the 5 propositions:
a : Something is.
b: Something always was.
c: The things which now are have risen from nothing.
d: It exists in the necessity of its own nature (i.e; the *something* spoken of above).
e: It exists by the will of another Being.

The program consists of the only rule

$$\begin{aligned}&\text{ValueOfSomethingHasAlwaysExisted}(b) \rightarrow,\\ &\quad \{a = \mathbf{1},\\ &\quad a \Rightarrow (b \vee c) \wedge \neg(b \wedge c),\\ &\quad a \Rightarrow (d \vee e) \wedge \neg(d \wedge e),\\ &\quad d \Rightarrow b,\\ &\quad e \Rightarrow \neg c\};\end{aligned}$$

To solve the problem we write the query

ValueOfSomethingHasAlwaysExisted(x)?

and we obtain the only reply

$$\{x = \mathbf{1}\}.$$

Here is an abbreviated trace of the calculation of this reply. The initial state is

$(\{x\}$, ValueOfSomethingHasAlwaysExisted(x), $\{\})$.

By applying the one rule, we progress to the state

$(\{x\},\ ,\{x{=}b, a = \mathbf{1}, a \Rightarrow (b \vee c) \wedge \neg(b \wedge c), a \Rightarrow (d \vee e) \wedge \neg(d \wedge e), d \Rightarrow b, e \Rightarrow \neg c\})$.

By eliminating the variables b and a we obtain

$(\{x\},\ ,\{(x \vee c) \wedge \neg(x \wedge c) = \mathbf{1}, (d \vee e) \wedge \neg(d \wedge e) = \mathbf{1}, d \Rightarrow x, e \Rightarrow \neg c\})$,

that is

$(\{x\},\ ,\{c = \neg x, e = \neg d, d \Rightarrow x, e \Rightarrow \neg c\})$.

By eliminating the variables c and e we obtain

$(\{x\},\ ,\{d \Rightarrow x, \neg d \Rightarrow x\})$

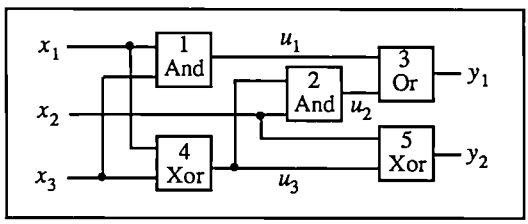
which simplifies to

$(\{x\},\ ,\ \{x = \mathbf{1}\})$.

Remember that the machine will not make the same simplifications as those we present for pedagogical reasons. It will use different ones, certainly more complicated, but they will also be more systematic (see end of article).

FAULT DETECTION

Here is a more complex problem, again related to Boolean algebra. It has been proposed in [7]. This time, we are aiming to detect one or more defective components in an adding circuit which calculates the binary sum of three bits x_1, x_2, x_3 in the form of a binary number composed of two bits: y_1y_2. As you can see below, the circuit is made up of 5 components numbered from 1 to 5: two *and* gates (marked And), one *or* gate (marked Or) and two *exclusive or* gates (marked Xor). We have also added three variables u_1, u_2, u_3 to represent the output from gates 1, 2 and 4.



We also introduce 5 more boolean variables p_i to indicate that "gate number i has broken down". If we adopt the hypothesis that at most one of the five components has broken down, the program connecting the values x_i, y_i and p_i is

Circuit(<x1,x2,x3>, <y1,y2>, <p1,p2,p3,p4,p5>) →
AtMostOneTrue(<p1,p2,p3,p4,p5>)
{¬p1 => (u1 ≡ x1∧x3),
¬p2 => (u2 ≡ x2∧u3),
¬p3 => (y1 ≡ u1∨u2),
¬p4 => (u3 ≡ ¬(x1≡x3)),
¬p5 => (y2 ≡ ¬(x2≡u3))};

AtMostOneTrue(P) → OrOnAtMostOneTrue(P,p);

OrOnAtMostOneTrue (<>, 0) →;
OrOnAtMostOneTrue (<p>•P, p∨q) → OrOnAtMostOneTrue (P, q), {p∧q = 0};

If the state of the circuit leads us to write the query

$$\text{Circuit}(\langle 1, 1, 0\rangle, \langle 0, 1\rangle, \langle p1, p2, p3, p4, p5\rangle)?$$

the diagnosis is that component number 4 has broken down:

$$\{p1 = 0,\ p2 = 0,\ p3 = 0, p4 = 1, p5 = 0\}.$$

If the state of the circuit leads us to write the query

$$\text{Circuit}(\langle 1, 0, 1\rangle, \langle 0, 0\rangle, \langle p1, p2, p3, p4, p5\rangle)?$$

the diagnosis is that either component number 1 or component number 3 has broken down:

$$\{p1 \vee p3 = 1, p1 \wedge p3 = 0, p2 = 0,\ p4 = 0, p5 = 0\}.$$

TREE MANIPULATIONS

Our last example illustrates Prolog III's capacity for refined tree manipulations without having to make use of external functions. The problem posed is the calculation of the list of leaves of a finite tree. Here is the program without any explanations.

```
LeavesOf(tree, leaves) → PlusLeavesOfTree(<>, tree, leaves);

PlusLeavesOfTree(leaves, label[<>], <label[<>]>•leaves) →;
PlusLeavesOfTree(leaves, label[list], leaves') →
    PlusLeavesOfList(leaves, list, leaves'),
    {list ≠ <>};

PlusLeavesOfList(leaves, <>, leaves) →;
PlusLeavesOfList(leaves , <tree>•list, leaves'') →
    PlusLeavesOfTree(leaves, tree, leaves')
    PlusLeavesOfList(leaves', list, leaves'');
```

The query

$$\text{LeavesOf}(\text{weights}(\text{"Joe"},< 80,\text{kilos}>,1),\textit{leaves})?$$

should produce the following reply

$$\{\textit{leaves} = <\text{'J'}, \text{'o'}, \text{'e'}, 80, \text{kilos}, 1>\}.$$

PRACTICAL REALISATION

I should like to finish this article with some information on our very first prototype for the Prolog III interpreter. It is written in C, apart from certain parts of the environment (by Pascal

Bouvier), which are written in Prolog II. In due course, these parts will be written directly in Prolog III. The most remarkable feature is the size of the constraint resolution program: 50 times bigger than a standard unification program!

The core of the Prolog III interpreter, designed by Touraïvane is essentially a non deterministic machine with two stacks whose elements are pushed and popped synchronously. In the first stack we create all the structures which represent the states through which we pass. In the second stack we store address-value pairs to record all the modifications we have made in the first stack, so as to be able to make the necessary restorations for "backtracking". A general garbarge collector removes from time to time the useless parts of the two stacks, while preserving their topographies. The core also solves most of the equations and constraints of type $\neq$. The algorithms used are basically extensions of those used in Prolog II and described in [5]. These extensions deal mainly with lists and numerical variables which are not constrained to be positive.

The core of the interpreter calls on two modules: one for the processing of Boolean algebra, the other for the processing of numerical constraints of type $\geq$. The module for Boolean algebra has been designed by Jean-Marc Boï and Frédéric Benhamou. The algorithms used are essentially those of Pierre Siegel [9]. On the one hand they check if a system of constraints can be satisfied, and on the other they simplify a system into a system which only makes use of a given subset of variables.

The arithmetical module, written by Michel Henrion, processes variables which are constrained to be non-negative (these variables are introduced to remove the constraints of type $\geq$). Fundamentally, it consists of the programming of George Dantzig's Simplex algorithm [6]. A subtle process deals with "degeneracies", that is with the appearance of unexpected zeros [1]. To this has been added a non trivial process to deal with the remaining constraints of type $\neq$. The module also includes basic subprograms for carrying out operations of addition and mutliplication with infinite precision, (that is, using fractions whose numerator and denominator can have variable length).

THE FUTURE

A commercial product such as a Prolog III interpreter or compiler should be available in two years time. As our examples have shown, its applications will be varied, and our experience of Prolog tells us that most of them will be surprising. However I see two dominant areas of interest. The presence of linear inequalities makes it possible to solve traditional problems encountered in operational research: minimization of costs, planning etc, but with much greater flexibility. The presence of complete Boolean algebra makes it possible to improve the

formulation of reasoning rules for expert systems. It is no longer necessary to limit their reasoning models to the scheme "if that and that, then this". Logical uncertainties like "this or that is true", and logical negations like "this is not true" can now be used. In fact, we should soon be in a position to know more about the possible applications for Prolog III. The next stage in our ESPRIT project is to use our interpreter prototype to program the PROMOTEX system for diagnosis of car engine breakdowns.

ACKNOLEDGEMENTS

I would like to thank the CEA and the Association AMEDIA for their financial help in this project. I also thank DEC which by donating an "External Research Grant" enabled us to acquire useful computer equipments. Finally I would like to thank the members of the Ministry for Research and Higher Education, who have supported research efforts connected with the project, within the framework of their Joint Research Programs "Tools for Artificial Intelligence" and "Artificial Intelligence".

REFERENCES

[1] Balinski Michel L. and Ralph E. Gomory, A Mutual Primal-Dual Simplex Method, *Recent Advances in Mathematical Programming*, Edited by R. Graves and P. Wolfe, McGraw-Hill, 1963.

[2] Boole George, *The Laws of Thought,* Dover Publication Inc., 1958.

[3] Colmerauer Alain, Henry Kanoui, Robert Pasero and Philippe Roussel, *Un système de communication homme-machine en français*, Research report, Groupe Intelligence Artificielle, University Aix-Marseille II, 1973.

[4] Colmerauer Alain, Prolog in 10 figures, *Communication of the ACM*, Volume 28, Number 12 , December 1985, 1296-1310.

[5] Colmerauer Alain, Equations and Inequations on Finite and Infinite Trees, Invited lecture, *Proceedings of the International Conference on Fifth Generation Computer Systems,* Tokyo, November 1984, 85-99.

[6] Dantzig Georges B., Linear Programming and Extensions, Princeton University Press 1963.

[7] Genesereth Michael R. and Matthew L. Ginsberg, Logic Programming, *Communications of the ACM,* Volume 28, Number 9, September 1985, 933-941.

[8] Robinson Alan, A machine-oriented logic based on the resolution principle, *Journal of the ACM*, 12 December 1965.

[9] Siegel Pierre, *Représentation et utilisation de la connaissances en calcul propositinnel*, Doctoral Thesis, Faculté des Sciences de Luminy, University Aix-Marseille II, July 1987.

Project No. 1063

AN EXPERIMENTAL PROTOCOL FOR THE ACQUISITION OF EXAMPLES FOR LEARNING

Jim Blythe, David Needham

GEC Research, West Hanningfield Rd., Chelmsford CM2 8HN, England.

Patrick Corsi

Cognitech, 167 rue du Chevaleret 75013 Paris, France.

The INSTIL project addresses the automation of knowledge acquisition for expert systems using techniques of machine learning. The important research contributions of this project lie in its approach to 'noise' in the input data to the learning system. This approach is based on the integration of three learning algorithms, each with a fundamentally different basis, that have been previously implemented and developed by the partners. The different strengths of these algorithms are able to complement each other.

Over the course of the project, the system is being tested in the field of plant pathology, although the range of application extends to that of classification expert systems. We present the test field as a case study in the use of the INSTIL system, focussing on the protocol used for gathering examples. These examples can be usefully described by non-experts, reducing the cost of knowledge acquisition and combatting one type of noise. We also characterise the problem domains for which the INSTIL system is well suited.

We generalise the experience of the case study to suggest key points for a protocol for knowledge acquisition in machine learning. This takes various aspects of noise into account, and allows the final representation language to be oriented towards the targeted users of the system, as well as the experts.

1. INTRODUCTION

The INSTIL project aims to build a learning system adapted for use in real-life environments. To that end, the system should be developed in conjunction with domain of expertise. The domain of plant disease diagnosis was chosen at the beginning of the project because of two main observations:

- Some machine learning and KBS implementations already existed in this field at the time (eg Soja [Michalski & Chilausky 81]) and more were on the way (eg Wheat Counsellor by ICI), and

- an expert system on tomato plant pathology had already been developed by INRA (Institut National de Recherche Agronomique, France) and Cognitech, thus allowing the project to be evaluated by direct comparison with an existing system at the conclusion of the project.

One problem that is still to be overcome in applying learning systems in such a domain is that of noise, which we take to mean incorrect or incomplete data in the training examples. Dealing with noise is the major work of the INSTIL project. The approach taken is based on the integration of three existing learning systems, all of which have been developed and used by the partners, and

which have different strengths and weaknesses [Smallman 85]. From them, we can produce a system for rule induction that combines a powerful representation with low sensitivity to noise.

However, we do not rely solely on the rule induction phase for dealing with noise in the process of learning. While a great deal of effort has recently been put in to the problems that noise causes for the machine learning task [Quinlan 86], [Niblett 87], most of this work has focussed on the rule construction phase, concentrating on making it less sensitive to individual examples that may be erroneous. Principal techniques involved in this task have been pruning [Briemann et al. 84] or truncation [Mozetic 86] of the rule base to make it less sensitive to detail.

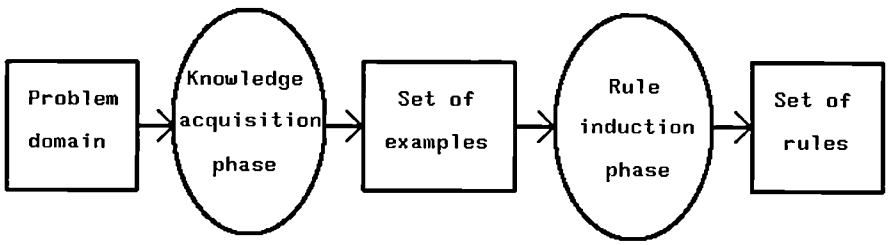


FIGURE 1

The overall process of constructing a knowledge base by machine learning.

We view the treatment of noise as a process that should affect every stage of the learning process, as shown in figure 1. A particularly important phase is that of example acquisition, which is the subject of this paper. We present the methodology that the project has developed for conducting knowledge acquisition in a noisy domain, and provide a case study in the domain of tomato pathology. This phase was carried-out between April and June 1987.

The idea of treating noise at the acquisition stage is not based on eliminating all noise from the examples: that would undermine the capability of the induction module to respond to the noise that is present in the domain both at consult time and learn time [Quinlan 86]. Our study aims rather at classifying the noise that will be present by source and effect, to maximise the information available to the learning system, and at reducing the noise that is only present during the acquisition phase.

1.1. Rationale for the acquisition of training examples

In the next section we develop some of the background necessary to construct a simple model of the process of example acquisition. Central to this work is the desire to produce guidelines for gathering examples that can help to combat noise in a domain that is subject to it. This will be necessary to ensure that the final system is well applicable to a range of problem domains. It is also essential in order to deal with noise in a coordinated way. To do this, we investigate the types of noise that can particularly affect the process of knowledge acquisition, and develop a simple model of the process.

One manifestation of the noise inherent in a domain is that *different people would describe the same event differently*, even if they use the same description language. For example, the threshold at which red becomes orange is not a

universal standard, nor is it clear when a leaf's edge stops being smooth to become jagged. This phenomenon seems to be strongly linked with the 'skill' level of the observer, since an expert can see much more detail in his domain than a novice.

This observation leads us to a major principle of our approach to knowledge acquisition: *that examples should be described by a number of people, preferably of the same level of skill in the domain as the targeted end user.* This is in contrast to the classical expert systems approach, where the skilled knowledge engineer derives the system from a few experts.

The work of *diagnosis* must of course be done by an expert, in order to capture his or her knowledge. Thus the scene is set for a number of active agents of differing skills and tasks in the knowledge acquisition phase. We next analyse the degree to which the different types of noise identified affect each agent.

Finally, we take into account the different types and sources of examples that may be used for rule induction. It is estimated that it would take seven years to see an example of every type of tomato disease that can occur in France. Thus a reasonable system must make use of examples taken from different sources, e.g. photographs and the memories of experts. These must be treated in different ways as described below, although this is mainly a problem for the learning system, and is outside the scope of this paper.

In section 3, we present some of our experiences of knowledge acquisition in the tomato plant domain. This is compared with the ideas developed in section 2. In the final section, we present partial results of the study, and discuss how the lessons learned might be generalised to apply to other domains.

2. THE ACQUISITION METHODOLOGY

2.1. Types of noise to be treated

We now come to look more closely at the kinds of noise that are present during the task. We are not concerned in this paper with their treatment after the acquisition phase is completed, or with other types of noise that may then arise. The reader is referred to [Kodratoff et al. 86] for the previous work on which this section is based. The consortium pinpointed 11 types of noise that have an impact on this phase, which are as follows. The number of + signs indicates increasing importance.

N1. A faulty description language:
Frequently a learning system fails because the description language for examples has not been rich enough to express the differences between them. The use of a pilot phase (see below), and the analysis of some examples by an expert help to alleviate this problem.

N2 Descriptors that are costly to check: +
Very often, descriptors are costly to check since:
(a) they necessitate killing the plant or require a lot of plants, or
(b) when a sample is not complete (consists only of parts of plants) it would cost time to obtain certain measurements.

N3. Descriptors hard to notice: +++
Very often, these descriptors are the most pertinent ones. They are also the ones the experts like to use for showing off (!). The eyes of the experts are accustomed to finding them, but not those of a technician; the expert can see signs that are hardly visible, and can recognise signs under a wide range of polymorphy.

N4. High level concepts similar and/or blurry: ++
Some diseases express similar symptoms under particular conditions. The expert himself is sometimes obliged to do laboratory tests. In fact, descriptors needed to differentiate such concepts are often costly or hard to notice.

N5. Randomness of natural phenomena: ++
Few natural domains are without some variation of features across the sample space. Any acquisition method will be at the mercy of factors beyond our control - this is one reason to make the operation as large as possible.

N6. Variation of tolerance of tests: +
May happen, particularly with accuracy of measurements (length of a plant).

N7. Mistake in providing a value: +
This will certainly be the case with non-naive users or plant specialists who may allow their views to be influenced by what they expect to see. As origins, we can mention (a) lack of attention, and (b) prejudiced view. When naive users make mistakes, they can often be corrected in the early stages of the experiments. The risk is decreased by on-line help and the spelling checker provided by an interactive questionnaire.

N8. Leaving out a descriptor: +++
This is inherent in the domain, because an example is often incomplete. Beyond noting that a descriptor cannot be evaluated, we feel it is not appropriate to treat this type of noise during the acquisition phase, since choosing 'good' examples will lead to a bias that is not present when the system is in use. This problem is handled during the learning phase, where the system chooses the more 'reliable' descriptors.

N9. Wrong class attributed to an example: ++
This case probably has the same origin as N7, that is, prejudiced view.

N10. Bad examples with too many symptoms: ++
That could happen because normal (not diseased) plants present imperfections that could be described by a non-specialist of the plant. It might happen also when a plant has multiple diseases where one (or more) diseases are not diagnosed.

N11. Bad or incomplete example space: +++
A law of the domain. In general, a set of training examples that span the problem domain may take several years to produce. This fact of plant pathology makes it very likely that, in the long run, a learning apprentice system capable of adapting while being used will be appropriate for the domain [Mitchell et al. 85].

In the next sections we develop the possible solutions we propose to minimise the effects of these types of noise. We stress again that treatment must occur throughout the learning process, and some types are more easily treated during the induction phase.

2.2. A typology of the domain actors.

This section is based both on the ideas mentioned in section 1, and on our practical experience. We differentiate three categories of personnel involved at some specific points during the search and collection of the examples. They are:

- The *Farmer* (F) is in charge of growing and curing the tomato plants. He often asks the Technician (see below) for help.

- The *Technician* (T) is knowledgeable about the tomato field, but sometimes has to ask the expert's help for a difficult diagnosis. We should distinguish here between the "naive" technician (NT), who is an occasional user but not trained or accustomed to the questionnaire, and the "non-naive" technician (NNT) who knows the questionnaire well in some form or another.

- The *Expert* (E) is the agreed reference for expertise in the application field. He is the one whose task is to validate the results at each step of the data flow. His diagnoses shall always be kept as a major reference as far as all future performance and validation tests are concerned.

The types of noise described in section 2.1 have an impact on each category of user in the ways described below.

- The Technician: N3, N7, N8 and N10. These probably depend on the technician's level of knowledge about the tomato. We may distinguish between the two types of technician:

  Naive Technician: *makes mistakes in answers*
  N3: he is not familiar with the descriptors in the questionnaire.
  N7: he will probably generate less of this noise, because of the close attention he must pay to complete the questionnaire.

  Non-naive Technician: *presents bias about knowledge*
  N3: he will probably generate less of this, because he is used to thinking in terms of our descriptors.

- The Expert gives incomplete description of the world
  N9: the source of noise we expect is the incomplete diagnosis.
  N4: probably comes from N5 or N2, or from a lack of laboratory tests.

2.3 Different types of 'example' to be used in a learning system.

There are two major types of examples that will be collected and used in a system - 'real' ones, that come from the field and are diagnosed by the expert, and 'synthetic' ones that must be used to supplement the base of training examples to provide an adequate coverage of information.

There are three subtypes in the category of real examples:

- Those cases which would normally come from the field to Technical Centres to be diagnosed. The example is described after reception and before diagnosis by an expert. Their acquisition is completely passive.

- Those cases that come to be known to technicians and must be collected. This is a slightly more active form of acquisition.

- Those cases that would otherwise remain unknown to the technicians. By introducing our work force into the field, under the guidance of technicians and other personnel, an active search can bring results. This is the most time-consuming process.

A mixture of the passive and active methods of receiving examples should be used in such a domain as this, if possible, to overcome any bias that may be introduced by using only one form of acquisition.

The 'synthetic' examples can be split into those taken from photographs, where certain descriptors often have to be guessed, those taken from written records of previous seasons, either on-line or not, and those that come from an expert's memory. In the work done so far, only photographs have been used as synthetic

examples.

It is important to remember that the synthetic examples have in some sense been singled out and kept as "case studies", often used for teaching purposes. In general, they are either proto-typical members of their class or highly atypical, kept for their strangeness. These factors are important in the design of a learning system capable of accepting both types of example, but do not concern us here, except to note it is important to differentiate them.

Items / Examples	Interest	Inconvenience	Noise
Natural examples	Actual farmers problem	- Non complete - Weather dependant - Statistically bad	N1 to N11
Expert memory	Completeness	Idealised examples	N8, N10
Descriptive papers	Case library	Other representation	N8, N2
Photographs	Case library	Partial representation	N8, N3

FIGURE 2
The types of examples.

3. A CASE STUDY IN THE TOMATO PLANT DOMAIN

As mentioned in section 1, the acquisition phase was completed in early Summer 1987. In France, INRA is widely recognised as the best organisation for conducting large-scale experiments in plant pathology. They provided the experts with whom Cognitech built the original TOM expert system, and also provide help in our current work. Within INRA, Avignon is of prime importance for research on tomato pathology involving top-level experts, trained technicians and several surrounding sites for growing tomato plants. For this reason we chose Avignon as our area for example acquisition.

Any individual case study of example acquisition will have slightly different circumstances, leading to a slightly different organisation of domain actors and a slightly different protocol. In this section, we describe the organisation of people in Avignon, and their interaction. In order to handle a large flow of examples, an automated questionnaire has been developed by Cognitech, providing a structured way to store examples, with help facilities about the description space. We briefly discuss this system and we also look at how the acquisition phase can be controlled and calibrated using a pilot phase.

3.1. Domain actors

The classes of people active in Avignon are the three mentioned in section 2.2. At the start of the acquisition phase, there were good relations already established between INRA and the farmers of the area. We introduced some people into the area as technicians, to collect examples actively for us, and they worked through these established relations. We note the following specific problems in the current domain:

- There is some loss of information between the farmer and the expert, because of a typical reluctance to call an expert in. We therefore lose a few examples that way.

- There is the possibility of information loss due to the presence of various hierarchy levels and some :old communications habits within the rural community, which the filler of the questionnaire may not always be accustomed to. This loss he has to detect.

- Finally, it seems better to describe the tomato at the site where it grows (when possible), in order to avoid forgetting some aspects or if an unavailable piece of information comes to be needed.

3.2. The Questionnaire

The questionnaire in use in Avignon is a software tool for knowledge acquisition that structures and formats the examples as they are entered in full screen mode. It was developed in Common Lisp on a SUN-3 and ported to a BULL DPS8/Multics system at INRA. Consultations of the questionnaire are made through TRANSPAC, the French public network.

The questionnaire contains a tree structure of menus, which match the hierarchy of objects in the domain knowledge. It allows the user to make comments about any descriptor, as well as enter the objects and values that are processed. When requested, it provides on-line help facilities and can give default values for descriptors. In order to help noise elimination at later stages, the questionnaire also makes a note of the date, time and user's identity. The diagnosis is entered separately from the questionnaire.

3.3. The control of the acquisition phase

We found it useful to carry out the acquisition process for two weeks before we began to accept the results, in order to remove the possibility of noise introduced by lack of familiarity with the system. We view this period, the 'pilot phase', as the control and tuning of the acquisition phase, which may serve these purposes:

- *validation* of the collection of the examples.

- *calibration* of the parameters of noise (costs, defaults etc.), to estimate the noise produced during the acquisition phase.

- the establishment of links with workers in the field and experts.

- eventually, the trace of this event enables us to propose a methodology of data acquisition for an INSTIL environment to generate a complete database.

The examples produced during this period were examined both by experts and technicians. Likely areas of vulnerability to noise were brought to light, and in addition several improvements to the description language being used were made, where it had been found to be inadequate. We note here that this control period inevitably introduced some bias in the tomato domain, because it took place at the start of the season, and all the plants were young, but we do not see how this could have been avoided without spending an entire season as a control phase.

4. RESULTS AND CONCLUSIONS

We are able to provide the final results of the acquisition phase covering the pilot phase and another two-and-a-half months of data collection. From our initial experience we make some suggestions about what a more general knowledge acquisition phase for an inductively generated knowledge based system might be like.

4.1. The diseases covered within the crop season.

During the acquisition phase we obtained 203 exploitable real examples and 105 exploitable synthetic examples (from photographs). They together provided 77 different diagnoses (46 simple diseases and 31 mixed diseases), covering most of the known diseases. Other examples were collected, but rated as unexploitable for various reasons (communications failure, lack of descriptors, etc).

The following table gives the diseases most commonly observed and diagnosed by the expert. The frequency numbers include also occurrences of the diseases in combination with others.

DISEASE	FREQUENCY
Botrytis	34
FOL	22
Phytotoxicite	22
Corky-root	21
Probleme alimentaire	21
Moelle noire	17
Verticilliose	14
Cladosporiose	12
Alternariose	11
Argenture	11
Carence en fer	11
Corynebacterium	11
Intumescences	10
Moucheture	10

4.2. A proposed methodology for example acquisition

Some of our experiences with the test domain are individual to that domain, but many can be generalised to form a set of guidelines for the task of example acquisition. We now summarise the work presented here by way of considering these guidelines.

- Domain actors
  In most domains, the target users will have a different way of describing the domain from the experts. This can be a problem in classical expert systems. We propose that examples should be described by people of the same skill level as the end users, working in the "field" with those users. The examples should be classified by the expert, however.

- Documentation
  The examples gathered should be carefully documented, in case they are later questioned by a learning system. In particular, it is useful to save the date and the questionnaire filler's identity. This is best done by questionnaire software that can also provide help about the description language.

- Pilot phase
  It will often be an advantage to spend a small percentage of the total time available in a 'dry run' or pilot phase. This allows the users to get familiar with the system, the experts to calibrate various types of noise that may be introduced, and provides an opportunity for the questionnaire to be modified if necessary. This phase should preferably involve some test runs with the learning system to examine the knowledge being formed.

This project has looked extensively at the possibility of using machine learning techniques to generate commercial expert systems. Most of the research work in this field uses examples bases that are easy to acquire, such as are provided by medical records [Quinlan 86], [Bratko et al. 87], [Michalski et al 86]. We have investigated the case where such bases are not so readily available.

We find that the task of acquisition can, indeed should, be shared between a number of technicians, rather than a few experts. Moreover, the presence of end-users in the acquisition process helps ensure that the domain language used in the final system is oriented as far as possible towards the end-user as well as the expert. It also seems a quicker and cheaper process than conventional knowledge elicitation for expert systems. Finally, the process of actively creating the example base allows us to introduce noise handling techniques into every stage of the rule induction process.

ACKNOWLEDGEMENTS

We are grateful to the INRA institute for providing the necessary human and material resources for the collection of examples. Special thanks to Christine Piaton and Noel Conruyt, students in agriculture, and to M. Dominique Blancard, expert in tomato plant pathology at INRA - Avignon. We also acknowledge the contributions to this work from Michel Manago and his colleagues at the Laboratoire de Recherche en Informatique, Universite de Paris-Sud, our partners in INSTIL, ESPRIT project P1063.

REFERENCES

Briemann et al. 84, Briemann, Friedman, Olshen & Stone, Classification and Regression Trees, Wadsworth Press, 1984.

Kodratoff et al. 86, Kodratoff, Y., Manago, M., Blythe, J., Smallman, C., Andro, T., "The Integration of Numeric and Symbolic Techniques in Learning", presented at the AAAI Workshop on Knowledge Acquisition, Banff, Canada, 1986

Mozetic, I. 86, "Knowledge Extraction through Learning from Examples", in Machine Learning: A Guide to Current Reasearch, Mitchell, Carbonell, Michalski, eds., Kluwer Academic Publishers, Boston, MA.

Michalski & Chilausky 81, "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction From Examples: A Case Study Involving Soybean Pathology", in Fuzzy Reasoning and its applications.

Michalski et al. 86, Michalski, R., Mozetic, I., Hong, J., Lavrac, N., "The AQ15 Inductive Learning System: An Overview and Experiments", Intelligent Systems Group, Dept. of Comp. Sci., University of Illinois.

Mitchell et al. 85, Mitchell, T., Mahadevan, S., Steinberg, L.,"LEAP: A Learning Apprentice for VLSI Design", in Proceedings of IJCAI-85, Los Angeles, August 1985.

Niblett 87, "Constructing Decision Trees in Noisy Domains", in Progress in Machine Learning, I. Bratko & N. Lavrac eds., Sigma Press 1987.

Quinlan 86, "The Effect of Noise on Concept Learning", in Machine Learning, volume 2, Michalski, Carbonell & Mitchell, eds., Morgan Kauffman 1986.

Smallman 85, Technical Annexe for ESPRIT project 1063: The Integration of Numeric and Symbolic Concepts in Learning, 1985.

Project No. 1133

A PRODUCTION RULE LANGUAGE FOR DATABASES EXTENDED TOWARDS THE SUPPORT OF COMPLEX DOMAINS *.

*Gerald Kiernan, Christophe de Maindreville, Eric Simon

INRIA Rocquencourt - BP 105
78153 Le Chesnay Cedex, France
*MASI Laboratory
University of PARIS VI

This paper presents (i) the integration of a production rule language in a DBMS and (ii) the support of complex domains within rules using a special purpose LISP language interpreter with a loose coupling strategy. A RDL1 production rule consists of a conditional part which is a relational calculus expression and of an action part which is a sequence of deletions and insertions in the database relations. This language is compiled in a new execution model based on Predicate Transition Network which permits an integration of the rule language in the DBMS. Abstract data types (ADT) are defined by the set of operations that can be performed on a domain data structure (DDS). Complex domains are ADT with operator inheritance based on a hierarchical "IS A" relationship between domains. All complex domains are represented as LISP structures. User defined operators are stored in function base. Efficient retrieval can be ensured using a clustering method which takes into consideration frequently applied functions.

1. INTRODUCTION

Existing Database Management Systems (DBMS) are inadequate for many new potential applications. Such applications have been studied in the Esprit Project 1133 and include a software engineering environment and an intelligent training system for helicopters[1]. The initial results [2] point to the need for including in the DBMS efficient handling of new domain types and more intentional knowledge defined in a flexible and modular way. New domain types may include sets, lists, trees, graphs, graphics, ... with associated operators. Intentional knowledge requires general production rules not restricted to a Horn clause form and that support multiple actions including updates. Furthermore, the applications call for a tight integration of the rules with the DBMS instead

* This research has been done in the framework of the ESPRIT Project ISIDE 1133.

of a loose or tight coupling. Recursive rules appear to be an application issue only under two conditions : (i) the rules must be general enough to support operations such as a student's evaluation in the training system or pattern matching in a software engineering environment and (ii) the recursive rules must not be too complex (for instance they remain linear i.e., the recursive predicate in a rule appears only once in the antecedent part of the rule). Under these two conditions, the intelligent training system is a typical application requiring many recursive rules. Another important issue is the need for similar performances when accessing either extentional or intentional knowledge.

From these initial considerations, the project researches two main problems. The first one concerns the design of a new knowledge representation formalism which supports the concepts of rule and object and permits an efficient integration with a relational DBMS (RDBMS). The second one is, assuming that massive amount of physical main memory will be available on forthcoming computers, to propose an efficient implementation of an extended algebraic machine for the knowledge representation level. This paper focuses on the first problem and a few ideas on the second problem are presented at the end of the paper.

The first research problem has been divided in two parts : how to integrate general rules with an RDBMS and how to implement complex domain definition and manipulation in an RDBMS. We first started with a study of the state of the art and then two propositions emerged. The first proposition led to the design of an original production rule language called RDL1. Using this language, two kinds of database relations are handled : *base* (also called extentional) relations whose tuples have been entered explicitly by the user and *derived* (also called intentional) relations whose tuples are either explicitly given or obtained dynamically with rules. A production rule in RDL1 consists of a conditional part which is a tuple relational calculus expression and a consequent part which is a sequence of insertions and deletions of tuples in (derived or base) database relations. More precisely, they are two *elementary actions*, denoted "+" and "-". The update action "+" takes a ground fact as argument and maps a database state into another state which contains this fact. Thus the action "+" inserts a tuple into a (derived or base) relation. On the contrary, the action "-" takes a fact and deletes it from a (derived or base) relation. Complex update actions can be specified in terms of these two primitive actions. *Parametrized and multiple updates* are defined using variables as arguments of actions. Finally, all these updates can be rewritten into a sequence called an *action*. Then, we provided a compilation technique for transforming the rules into a new execution model based on Predicate Transition Networks, called a Production Compilation Network (PCN). This model is not only a static graphical representation of the connections that exist between rules and predicates but also a tool used to dynamically control the execution of a rule program. In particular, a PCN serves as a basis for a tight integration between RDL1 and the relational DBMS.

The second proposition is the "on the side" extension of an RDBMS with a special purpose LISP language interpreter designed as an integrated DBMS processor. This type of solution was introduced for the INGRES system in [3], [4, 5] and can be contrasted to an "on the top" approach

as described in [6,7,8] where an interface is created to simulate a new or extended data model. Our approach uses the rich typing capability offered by abstract data types (ADT). Abstract data types are defined by a domain data structure (DDS) with its associated operators and are implemented as LISP functions. Complex domains are ADTs with operator inheritance based on a hierarchical "IS A" relationship between domains. All complex domains are represented as LISP structures. New user defined functions are then stored in the database. The user can reference new complex domains when creating new relations. He can also define new functions for these domains and use them in an extended version of SQL. Complex domains can appear in any part of a relational query (projection, restriction, join). For example, the new complex domain DIMENSIONS can be created. Consider the relation RECTANGLE (NO:integer, DESCRIPTION:texte, SIDES:*dimensions*). The surface operator, defined for domain DIMENSIONS returns the surface value for rectangles. The SURFACE operator can also be implemented for other domains. The system will automatically choose the correct operator depending on the domain type. Efficient retrieval can be implemented using a clustering method which takes into consideration frequently applied functions. Moreover, intelligent filtering can be performed. For instance, tuples can be clustered according to the surface value of rectangles for efficient retrieval with queries applying a restriction based on this characteristic.

A first attempt at combining the two previous proposals has been investigated and the results are reported in this paper. Our approach is extending the rule based language in such a way that rules operate over relations defined with complex domains. The paper is thus organized as follows apart from this introduction. Section 2 presents the concept of complex domains and the language used to define these domains. The next section is devoted to the RDL1 rule language extended with complex domains. Section 4 presents a functional architecture of the proposed system and surveys query optimization techniques used to evaluate rule programs. Section 5 provides a qualitative analysis of the proposition herein presented. We outline the main limitations of this approach with respect to the goals assigned by the study of the potential applications. Performance issues related to the proposed approach are briefly discussed. Finally, future research perspectives are envisioned from this intermediate stage. Section 6 is the conclusion of the paper.

2. FDL1 : A FUNCTIONAL DOMAIN DEFINITION LANGUAGE.

2.1 Complex domains as abstract data types

The objective in extending a relational DBMS to a generalized one is to satisfy the requirements of new applications. The objects represented in these applications lose too much of their semantics when normalized. Most of the literature on generalized databases deals with the issue of complex object modelling. In programming languages, complex objects are defined as the potential behaviors of an object [9]. In database systems, complex object modelling is capturing the structure of an object in its totality or in its parts as represented in the tuples of normalized relations[10].

A *complex domain* can be defined in three steps: 1) the description of the *domain data structure* (DDS), 2) the definition of the associated *operators*, and 3) operator inheritance. The current notion of abstract data type (ADT) [3], [11] corresponds to the first two steps. New DDS capabilities should include hierarchical structures, sets, lists, ... Domains should be organized into a "ISA" hierarchy for operators inheritance. Futhermore, it should be possible to implement operators using previously defined ones. In "on the side" extensions of relational DBMS, a programming language is used to code new user-defined operators and domains. A user-defined operator can be applied on a per tuple basis (calculation) or it can be applied on a set of tuples (aggregate). The procedures (also called user programs) which implement the operators and domains are registered with the DBMS. When user-defined domains and operators are referenced in queries, the DBMS runs the appropriate procedure to resolve the query. An ADT is thus user-implemented and user-defined. The only knowledge that the DBMS has about the ADT is its name and the names of the operators which manipulate it. The knowledge about the ADT semantics is locked within the user code. When a query is processed, the task the DBMS has to accomplish is recognizing whether a domain or operator is user-defined or standard (i.e., predefined domains such as integer, real, text). If the domain or operator is user-defined, the system has to retrieve the code which implements it, link it with the DBMS program, run the user program and pass parameters between the DBMS program and the user program. Current implementations of ADT differ on (i) whether or not new operators can be defined for standard domains, (ii) the choice of programming language used to implement user-defined domains, (iii) whether the programming language is compiled or interpreted and (iv) physical access methods for ADT.

For instance, ADT-INGRES uses a compiled version of the C programming language to implement ADT. Due to the compiled environment, errors arising from user programs cannot be easily handled. User code is stored in sequential files and must be dynamically or statically linked to the DBMS program. In an interpreted version of the same language, linking would only require loading the user code into the interpreter. When registering a new domain or operator with ADT-INGRES, the user supplies the name of the operator, the number of bytes needed for both the internal representation and the external representation, and the name of the sequential file where the code is stored. In the case of operators, operator parameters and their types must be given. Because of the need for specifying the size of the physical representations of an ADT, it is difficult to implement type generators such as sets and lists which can be of variable size. A similar implementation of ADT can be found in [11]. In this one, in addition to calculations and aggregates there is another category of operators which are defined as transformation operators. These operators take a relation as input and return a relation as a result.

We defined a similar implementation strategy as ADT-INGRES but with different implementation choices. In our system, an object oriented version of the LISP language [12] is used to implement user-defined domains and operators. The environment is interpreted. All user code is stored in specific database relations called the function base. No secondary sequential files are used. User

domains are described using a hierarchical "IS A" relationship between domains for operator inheritance. The LISP language provides easy implementation for type generators such as sets, lists and trees. Special LISP functions are registered to verify domain values.

2.2 Syntactic definition of complex domains

The FDL1 language is used for the definition and manipulation of complex domains. Complex domains can only be manipulated through functions. It is thus necessary for the language to adhere to the functional approach. This is what motivated the choice of LISP. The functional approach meets the objective of being a general solution therefore, addressing a set of problems related to data representation and manipulation. The LISP processor is an entry point from which complex domains are registered with the DBMS. The interpreted environment ensures better control over programming errors in user-defined complex domains which might cause the DBMS program to terminate abnormally in a compiled environment. For example, a division by zero in a user function would be detected by the interpreter and control would be transferred to the DBMS program. This is much harder to accomplish if the user function is compiled. Also, compiled code must be statically or dynamically linked to the DBMS program to be run which presents an added difficulty.

LISP was initially designed as a type free language. This feature is not compatible with the consistency requirements of a DBMS. The *type* notion and *type hierarchy* notion were hence added to the language environment. The latter is for function inheritance among types. The type notion transposed to the DBMS environment describes DDS and their associated operators. Operator inheritance from generic domains is possible through the transposition of the type hierarchy. The Define Domain function (DD) is used to create a new complex domain. The general syntax of the DD function is the following:

(DD <generic domain><domain name> (<domain param>)(<domain definition>))

The "domain name" is the name of the new complex domain that is being defined by the DD function. The "generic domain" and its ancestors are the complex domains from which this domain will inherit operators. The "domain param" is an argument to which specific values will be individually bound and verified as belonging to this domain. The "domain definition" contains the predicates that the value represented by the "domain param" must verify to qualify as a particular occurrence of this domain. The DD function is thus a boolean function.

Example :

Rectangular shapes can be represented by the DIMENSIONS function:

```
(DD LISP DIMENSIONS (X)
 (and (listp x)
  (numberp (car x))
  (numberp (car (cdr x)))
```

```
(null (cdr (cdr x)))))
```

A value must be a list of two numbers to qualify as an occurrence of DIMENSIONS. DIMENSIONS is declared as a specialization of LISP. Therefore, all the basis LISP functions can be applied to attributes of complex domain DIMENSIONS. Besides being the name of the language, LISP is also the name of a complex domain. Associated to the LISP complex domain are all the basic LISP functions.[]

Operators for complex domains can be defined using the DE or the DF function. Contrary to the DD function, the DE and the DF functions are standard LISP functions, although their syntax has been modified in our version of the interpreter. The general syntax of these two functions is the same. We will simply state it for the DE function :

(DE <result type> <function name> (<domain name> (<arg list>) [<domain name>(<arg list>)]*)
    (<function definition>))

The "function name" is the name of the new function being defined by the DE or the DF function. The "result type" specifies the name of a simple or complex domain which represents the type of value returned by the function. The "domain name" is the type of the values in the "arg list". The "arg list" represents the function parameters. Thus, the function parameters can be from different domains. The "function definition" is the body of the function and contains the code which implements the function.

Example :
The following function defines the surface value for argument X of complex domain DIMENSIONS. The value returned by the function is from the predefined domain of integers.

```
(DE INTEGER SURFACE (DIMENSIONS (X))
(* (car x) (car (cdr x))))
```

The SQUARE function returns the BOOLEAN value TRUE if the argument X of domain DIMENSIONS is a square, otherwise FALSE.

```
(DE BOOLEAN SQUARE (DIMENSIONS (X))
(= (car x) (car (cdr x))))
```

The LENGTH and WIDTH functions for arguments of complex domain DIMENSIONS return an INTEGER value corresponding respectively to the length and width of a rectangle.

```
(DE INTEGER LENGTH (DIMENSIONS (X))
    (car x))
```

```
(DE INTEGER WIDTH (DIMENSIONS (X))
    (car (cdr x)))
```

To use complex domains in queries, the complex domains have to have been previously registered with the DBMS. Registering a complex domain simply implies storing its corresponding functions in the function base, the DBMS takes care of the rest. The function base is used to store user-defined functions and is accessed from within the LISP environment using special LISP functions to register, modify or delete specific entries. In an assertional language environment, the user can use complex domains in any clause of a query (projection, selection, join, aggregates). A function F applied to an attribute A of a relation R will be written using dot notation as R.A.F. An extended version of SQL based on these ideas was presented in [13].

3. THE PRODUCTION RULE LANGUAGE

The purpose of this section is to present an overview of the production rule language we use. The reader interested by a more detailed and formal definition of the rule language RDL1 is referred to [14, 15].

3.1. A tuple relational calculus

We first recall the basic notions of relational databases. A *relational schema* R is a finite set of attributes $\{A_1, ..., A_n\}$. Let dom (A_i) be the domain of values of attribute A_i. Note that the domains A_i can be defined as complex domains through the FDL1 language. A *constant tuple* $t = (c_1, ..., c_n)$ over a relational schema R is a mapping from R into dom $(A_1) \cup$ dom $(A_2) \cup ... \cup$ dom (A_n) such that for each i in $\{1, ..., n\}$ $c_i \in$ dom (A_i). We denote by dom'(R) the set of all possible constant tuples over R. An *instance* of a relational schema R, (sometimes called a relation), is a finite set of constant tuples over R. A *database schema* is a finite set of relational schemas. Finally, an *instance* I over a database schema S is a total function from S such that for each R in S, I(R) is an instance over R.

We use an extended version of the relational calculus of [16] based on a multi-sort logic where terms are typed. This relational calculus has two types of predicate symbols : The relational predicates (unary predicates) are those for which an interpretation corresponds to the instances of the relational schemas and allow the user to define the type of a tuple variable which ranges over a relation. For example, RECTANGLES is a relational predicate name. The non relational predicates correspond to the usual comparison ones (EQUAL_TO, GREATHER_THAN, ...).

A formula is composed of a first part which indicates the type of some tuple variables (the range definition part) and of a second part conjunctively connected to the former (called a *sub-formula*) which states a condition that must be satisfied by the tuple variables. For example, RECTANGLES

(x) indicates that x is a tuple variable ranging over the tuples of the relation RECTANGLES. Particular cases arise when the sub-formula is the predicate TRUE or when it does not contain any variable. In the following, we impose that our formulae meet the *range restricted property* [17] initially defined for closed formulae and that we extended to opened formulae.

We come now to the interpretation of a formula. Let D be the finite set of sorts composed by all the domains of attributes defined over a database schema S and by the domain of each relational schema. Thus, D = {dom (A_1), dom (A_2), ... , dom (R1), dom (R2), ...}. A database DB is the set of instances of all the relational predicates. We call DB an interpretation. Thus, DB = {I (R_1), I (R_2), ... }. Then, a formula is evaluated in the standard way.

3.2.The rule language.

A rule in RDL1 is not a logical formula put in a clausal form but is more in the spirit of productions in languages such as OPS5 [18] or other forward chaining rule based languages. The general form of a rule is the sentence : Condition → Action. A *condition* over a database schema S is a formula of the relational calculus. There are two *elementary actions*, denoted "+" and "-". The update action "+" takes a ground fact (i.e., a constant tuple) and maps a database state into another state which contains this fact. Thus, the action "+" inserts a constant tuple in a relation. For instance, + ANCESTOR (Asc = Bill, Desc = Marie) is a positive action. On the contrary, the action "-" takes a fact and deletes it from a relation. For instance, - ANCESTOR (Asc = Bill, Desc = Marie) is a negative action. Complex update actions can be specified in terms of a sequence of these two primitive actions. For instance, + ANCESTOR (Asc = Bill, Desc = Marie) - ANCESTOR (Asc = John, Desc = Peter) is a complex action composed with two elementary actions. Parametrized and multiple updates are defined using variables as arguments of actions. For example, + ANCESTOR (Asc = x.Father, Desc = x.Child) is a *parametrized action*, where x is a tuple variable ranging on a relation having the same domain as the ANCESTOR relation.

We impose the rules to be strongly safe rules as defined in [19].

Examples:

PARENTS (x) → + ANCESTOR (x) is a rule.

PARENTS (x) → + ANCESTOR (y) is <u>not</u> a rule because y does not appear in the condition part.

EMPLOYEE (x) AND x.Name = Jules → - EMPLOYEE (Name = Jules, Dept. = toys),
+ EMPLOYEE (Name = Jules, Dept = sports) is a rule.

A finite set of rules defines a *rule program* .

The semantics given to a rule defines a mapping over the database instances. Intuitively, given a database state I, this mapping is the set of immediate consequences of I using the rule that is the set of database states that are reachable by applying the rule only once. We shall now provide the details of the mapping. First, an elementary action takes a database state and maps it to another state

as follows : Given a database state I, an action + R (t), where R is a relational schema and t is a constant tuple, maps I to J such that :

$J(R) = I(R) \cup \{t\}$ and $J(T) = I(T)$ for each T such that $T \neq R$.

The action - R (t) maps I to J such that :

$J(R) = I(R) - \{t\}$ and $J(T) = I(T)$ for each T such that $T \neq R$

Example :

The semantics of the elementary action + ANCESTOR (Asc = Bill, Desc = Marie) implies adding the constant tuple (Asc = Bill, Desc = Marie) into the ANCESTOR relation and all the other relations remain unchanged.

The semantics of a general action is now given. We assume that all the variables contained in the action have been previously instantiated.

Given a database state I, the instantiated action $A = A_1 A_2 \ldots A_n$ maps from a state I to a state J such that for each relational predicate name R appearing in A :

$$J(R) = [(I(R) \cup I(R^+)) - I(R^-)] \cup [I(R) \cap I(R^+) \cap I(R^-)]$$

where $I(R^+)$ is the set of all the constant tuples t appearing in the elementary actions "+ R (t)" and $I(R^-)$ is the set of all the constant tuples t appearing in the elementary actions "- R (t)".

A feature of the language is that the interpretation of the action part of a rule is not performed as a sequential execution of insertions and deletions. Rather, a valued action is seen as an *atomic database update*. This property has several consequences. First, it guarantees that the order of insertions and deletions in the action part is irrelevant. Second, it nullifies an action which inserts and deletes the same constant tuple.

Examples :

Let P {A} be a relational schema, then

P (x) AND x.A = a $\rightarrow$ + P (A = a) - P (A = a) has no effect on P

P (x) AND x.A = a $\rightarrow$ + P (A = a) - P (A = a) + P (A = a) + P (A = a) has no effect on P.

P (x) AND x.A = a $\rightarrow$ + P (A = b) - P (A = b) has no effect on P.

P (x) AND x.A = a $\rightarrow$ + P (A = b) - P (x) changes the value of attribute A from a to b.

Finally, the semantics of a general rule is : for one condition of the rule which is interpreted to true when all the variables of the condition are replaced by constant tuples, execute the corresponding instantiated action over the database.

For one rule a *stable state* is reached when either (i) every instantiated condition of the rule takes a

false value or (ii) when the instantiated condition is true no new database state can be produced using the instantiated action. When a rule reaches a stable state for any initial database state, the rule is said to admit a *fixpoint*. If the rule admits a fixpoint and reaches a unique stable state then the rule is said to be *determinist*. This notion of stable state is easily generalized for a set of rules.

Examples :

Let R {int : integer} and S {A : integer, B : integer} be two relational schemas.
R (x) → + R (Int = x.Int + 1) has a infinite stable state,
R (x) and NOT Q (x) → + Q (Int = x.Int) has a finite stable state,
S (x) → + S (A = x.B, B = x.A) - S (x) has not a stable state,
R (x) and R (y) → - R (x) - R (y) + R (Int = x.Int + y.Int) has a finite stable state.

The semantics of a rule program is now defined as repeating the following procedure until a stable state is reached : (i) compute all the rules that can be fired i.e., whose condition takes a true value for a particular instantiation of its free variables, (ii) choose one instantiated rule among this set of firable rules and execute it, and (iii) return to step (i). Remark that the notion of *stable state* of an RDL1 program corresponds to the notion of model (or fixpoint) in logic programming. Starting from this definition, it is clear that an RDL1 program might not have a unique stable state or even one stable state. Syntaxic restrictions are presented in [14] which give sufficient conditions for a RDL1 program to have a unique stable state. In this case, a program is said to be determinist.

When a set of rules is declared, it is analyzed and compiled into an internal execution model. During this analysis, the system keeps all the possibilities opened for ordering the rules. Thus, the execution strategy is not *pre-compiled*. The system decides at run time only in which order the firable rules computed at the step (i) of the above procedure have to be fired. This point will be further discussed in the next section.

We conclude the presentation of the language with some examples of rule programs. Each program is introduced under the form of a rule *module*. A module takes input relations which are either base or derived relations and produces as output *target* relations which are defined using a set of rules.*Temporary* relations can also be used to compute intermediate results in a module.

Examples :

Consider the following problem about Rectangles (see section 2). Assume that for each rectangle in the RECTANGLES relation, we wish to write an RDL1 program to compute the derived relation RECT_PLUS = {N° : integer, Colour : text, Sides: Dimensions, is_greater_than: Set_of_rectangles} which represents for a rectangle, the set of rectangles which have individual surface value less than the surface value of this rectangle. The surface and GT (abbreviating greater than) operators and the set_of_rectangles complex domain have been previously defined for

rectangles. The following RDL1 program computes the RECT_PLUS relation.

```
MODULE : RECTANGLES ;
 target :
 RECT_PLUS ( N° : integer, Colour : text, Side: Dimension, is_greater_than: Set_of_rectangles )
 begin
 RECTANGLES (x) →+ RECT_PLUS (x, is_greater_than = ∅) ,
 RECTANGLES (x) AND RECT_PLUS (y) AND (x.Sides.Surface < y.Sides.Surface)
     →- RECT_PLUS (y)  + RECT_PLUS ( y, y. is_greater_than.union (x.N°))
 end.
```

The second example defines a target relation ANCESTOR from the PARENT (Father, Mother, Child) relation. With this standard *flat* schema, if a couple of persons has more than one child, the tuples will have to be repeated a number of times equal to the number of children had by couple. For example, let us consider the following instance of the PARENT relation :

Father	Mother	Child
Peter	Mary	Louis
Peter	Mary	John
Philippe	Louise	Mary
Philippe	Louise	Anne

This problem can be overcome if attribute Child is changed to a complex domain representing the set of children of a couple. Hence the new relation becomes :

Father	Mother	Children
Peter	Mary	{Louis, John}
Philippe	Louise	{Mary, Anne}

The rule module which solves this new ancestor problem is the following :

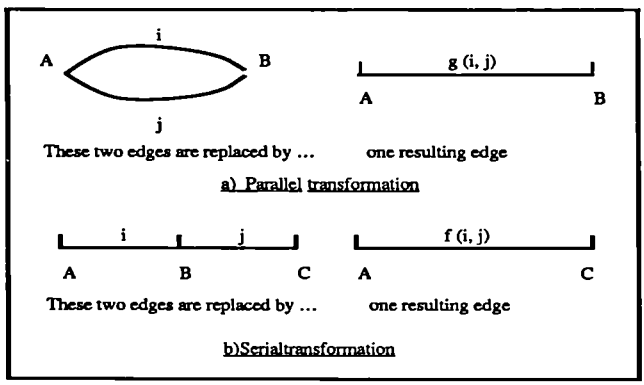
```
MODULE ANCESTOR ;
  target
  ANCESTOR (Father : text, Mother : text , Desc : set_of_text) ;
  begin
   PARENT (x) → + ANCESTOR (x),
  PARENT (x) AND ANCESTOR (y) AND ((x.mother.member (y.Children) = true) OR
       (x.father.member  (y.Children) = true))  → - ANCESTOR (y) + ANCESTOR (Father =
        y.Father, Mother = y.Mother, Desc = y.children.union (x.Children));
  end.
```

In this example, we find the two operators **member** and **union** defined for the *set_of_text* domain.

The member operator takes two parameters : the first one is the name of the parent that is to be identified in the second parameter which is the set of children. The union operator builds the set of children based on the union of both sets passed as parameters. For instance, after the firing of the previous module the instance of the ANCESTOR relation is as follows :

Father	Mother	Desc
Peter	Mary	{Louis, John}
Philippe	Louise	{Mary, Anne, Louis, John}

Now, in the third example, let us assume a base relation EDGE (W# : integer, Origin : char, Ext. : char, Label : integer), which represents oriented arcs in a graph. We can compute the "reduction" of the EDGE relation into the RESULT relation with the following rule module. The first rule copies all the tuples of EDGE into EDGE* and marks this operation by memorizing them into DONE. The second rule replaces two serial edges x and y from EDGE* by one new resulting edge whose label is a certain function f applied to the labels of x and y.



The third rule replaces two parallel edges x and y by one resulting edge whose label is a function g applied to the labels of x and y. Finally, the last rule computes the RESULT relation from the EDGE* relation. The diagrams above illustrate the application of the second and third rule. Intuitively, this module can be used to compute the impedance of a simple electrical circuit.

MODULE REDUCTION
target
RESULT (ORIGIN : char ; EXT : char, LABEL : integer),

EDGE (x) AND NOT DONE (x) → + EDGE* (x) + DONE (x)
EDGE* (x) AND EDGE* (y) AND x.Ext = y.Orig AND (∀ z ∈ EDGE*) [(z.W# ≠ y.W#) OR
(z.Origin ≠ x.Ext AND z.Ext ≠ x.Ext)]
→ - EDGE* (x) - EDGE* (y), + EDGE* (W# = x.W#, Origin = x.Origin,

```
   Ext = y.Ext, label = f (x.label, y.label)),
EDGE* (x) AND EDGE* (y) AND x.Origin = y.Origin  AND  x.Ext = y.Ext
                                AND x.W# ≠ y.W#
                        → + EDGE* (W# = x.W#, Origin = x.Origin, Ext = y.Ext,
                                        Label = g (x.label, y.label)) - EDGE* (x), - EDGE* (y),
EDGE* (x) → + RESULT (Origin = x.Origin, Ext = x.Ext, Label = x.Label) ;
 end.
```

4. QUERY PROCESSING TECHNIQUES

4.1. Definition of the Production Compilation Network

As pointed out in section 2, we follow a compilation approach for transforming a rule program into an execution model which serves as a basis for query processing. More precisely, the analysis of an RDL1 rule program is performed by the following steps. First, a syntactic and semantics analysis of the rules is performed. The next step transforms the rule program into an execution model based on Predicate Transition Nets. The third step performs some consistency checks over the net. Finally, the last step stores the net in a pre-compiled form in the rule base by incrementally updating the already existing PrTN. In this section, we just describe the model of PrTN that we use. The reader interested in the other phases is referred to [20].

Predicate Transition Nets (PrTN) derive from Petri nets. The reader is referred to [21] for a detailed presentation of the theory and modelling power of Petri Nets. The main difference with Petri nets consists in the fact that tokens are structured objects carrying values similar to database tuples and that transition firing can be controlled by imposing conditions over the token values. A formal definition of PrTN can be found in [22].

PrTN have been shown to be a powerful tool for modeling first order logic programs in Expert Systems [23] or for operational specification of process control systems. Indeed, a theorem in [22] states that each well formed formula in first order logic can be represented in a PrTN by means of a set of dead transitions (i.e., without transition's inscription). In this paper, we use a model, called Production Compilation Network (PCN), which is derived from the original PrTN in order to model the behaviors of RDL1 programs.

A PCN model has two aspects : the structure and the execution. These are also respectively called static and dynamic aspects. The structure aspect represents the interelationship between rules and relational predicates as specified by a rule program. The following associations can be made between rules and the PCN structure. We are able to set up a complete isomorphism between an RDL1 program and a PCN structure which is illustrated by the table in figure1 .

RDL.1	PCN
RULE	TRANSITION T
RELATIONAL PREDICATE P	PLACE P
Free variables $X_1,\ldots, X_n$ ranging positively over P	ARC (P,T) labelled by : $+X_1 +\ldots +X_n$
Bound variables ranging over P or free variables ranging negatively over P	ARC (P,T) labelled by : (.)
ACTION + P (t)	ARC (T,P) labelled by + t
ACTION - P (t)	ARC (T,P) labelled by - t
Sub-formula and negative range predicates	Transition's inscription

Figure 1: Correspondence table between rules and the PCN structure

An example of a PCN built from a set of rules will now be given. Figure 3 represents the RDL1 ANCESTOR module of section 2. Two relational predicates appear in the rules. They lead to the places PARENT and ANCESTOR. The first rule PARENT (x) → + ANCESTOR (x) is modeled by the transition t1 whose inscription is the sub-formula TRUE. The second rule is modelled by the transition t2, whose inscription is the sub-formula (x.mother.**member** (y.children) = true) OR (x.father.**member** (y.children) = true). The arc going from t2 to ANCESTOR is labelled by + (Father = y.Father, Mother = y.Mother, Desc = (y.Children.**union** (x.Children)).

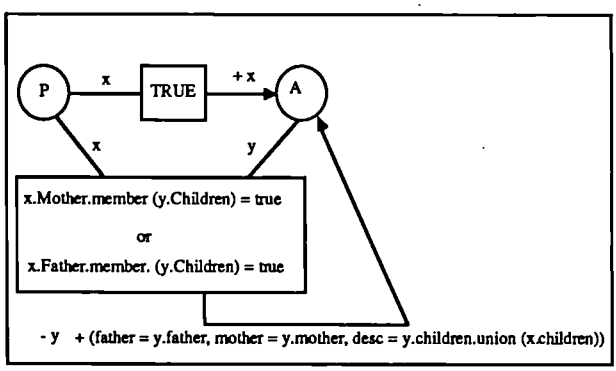


Figure 2: PCN for the Ancestor rules

The second example of PCN represents the module REDUCTION given in RDL1 in section 2.

Four relational predicates appear in the rules. They lead to the places EDGE (noted E), EDGE* (noted E*), DONE (noted D) and RESULT (noted R). The first rule is modelled by the transition t1 whose inscription is the negative range predicate NOT DONE (x). The outgoing arc from D is labelled by (.) since there is no free variable flowing on the arc. Each of the transitions t2 and t3 have one positive arc and one negative arc. Note that the presence of a quantified variable z in the transition's inscription of t3 does not lead to a label (.) on the input arc of t3 since tuple variables already appear on the label of this arc.

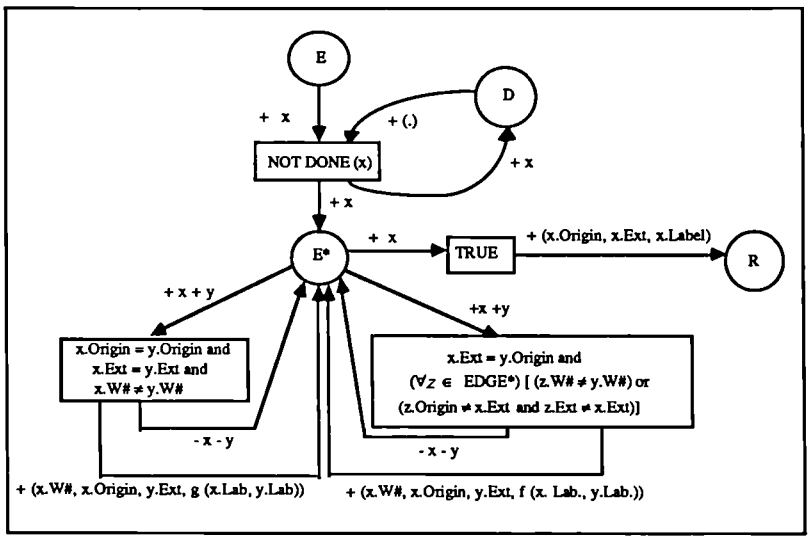


Figure 3: PCN for the module Reduction

4.2. Semantics of a PCN

In this section, precise definitions of the dynamic aspect of a PCN are presented. It is first defined by the notion of marking. A *marking* is a distribution of tokens over the places of a PCN. An *initial marking* consists of initializing a PCN with a particular marking. Two kinds of initial marking, called base and source markings, are distinguished : a *base (resp. source) marking* corresponds to a marking of only the base (resp. source) relations. The token is a basic concept in all Petri Net based models. In a PCN, a token represents a database tuple. A Petri Net based model executes by firing transitions. A transition can be fired if the transition is enabled. The execution rules of a PCN with respect to a PrTN are different in several ways that are specified below.

A transition T is *enabled* whenever the three following conditions are satisfied.

(i) Each input place P of T contains at least as many tokens as specified by the number of tuples figuring on the label of the arc (P, T). A tuple of the form (.) counts for zero.

(ii)Tokens occurring in the input places of T have values satisfying the transition condition.

We use a restricted semantics of PrTN. In fact, in all the applications, it makes no sense to draw the same tuple twice using the same rule. Therefore, we impose the capacity of each place, that is the number of copies of the same token a place can carry at the same time, to be bound to 1. A *duplication free* PCN is a PCN where two tokens of equal value in the same place are merged into a single token, and where a transition is not enabled if it can only produce into its output places already existing tokens.

When a transition T is enabled it can be *fired*. Firing T produces several actions. First, it duplicates from each input place P of T a number of tokens equal to the number of positive symbols labeling the arcs (P, T). Then, for each output place P' of T, it adds the tokens specified by the symbols occurring on the label of the positive arcs (T, P') and removes from P' the tokens specified by the symbols occurring on the label of the negative arcs (T, P'). This is a difference with the standard execution rule of a PrTN where all the tokens flowing through the input arcs of a transition T are consumed when the transition is fired. Thus, firing a transition T in a PrTn changes the state of both input and output places of T. Rather, we use conservative nets which where firing a transition T will only change the state of the output places of T.

At a given time, several transitions can be enabled. A *transition firing occurrence* is a couple (T, list) where T is a transition and where list is a list of tokens of the form $(x_1/a_1, \ldots, x_n/a_n)$ where the x_i are all the free variables figuring on the labels of the input arcs of transition T and the a_i are some tokens issued from the input places of transition T. The set of all transition firing occurrences for a given marking of the net is called the *transition's conflict set*. Note that for a single transition T, different lists of tokens can be used to fire the transition. Each one of these lists leads to an element $(T, list_i)$ in the transition's conflict set. We note Rel (T) (for Relevant set of tokens), the set of all the lists of tokens that appear in the transition's conflict set with the transition T.

Therefore, a PCN evaluator has to execute the following procedure portrayed on figure 6.

```
procedure evaluate (PCN, Initial marking)
    M ← Initial marking
    repeat
      compute the conflict set ,
      select a transition T in the conflict set,
      compute M = Reachable marking from M by firing T.
    until a halting condition is met.
```

Figure 4 : A PCN evaluator procedure

This procedure eventually terminates if a halting condition is met. This means that a stable marking is obtained for the net. The notion of stability is described below. For a given initial marking, a place P has a *stable marking* iff none of its input transitions is enabled. This notion can be used to

define the stability of a transition. A transition T is said to be *stable* for an initial marking iff all its output places have a stable marking. Then, a PCN has a *stable marking* for an initial marking iff all its transitions are stable. When the PCN reaches a stable state no transition is enabled and the PCN evaluator procedure halts. This corresponds to a fixpoint for the set of rules modelled by the net.

Another way for studying PCN is to focus not on what markings are reachable but on how they are reached. In particular, we are interested in the sequences of firing operations which lead from one state to another. Such a sequence is a sentence in the language associated with the marked PCN. Here, the purpose of this language is just to describe the flow of control in the query processing strategies in a clear and concise manner. The basic symbol of the language is $F_{t,\ list}$ which means that transition t is fired using the specified list of tokens. The symbol $F_{t,\ Rel\ (t)}$ means that the transition t is successively fired using the lists of tokens contained in Rel (t). The symbol σF_t means that the transition t is fired up to saturation without specifying the order in which the tokens are used to fire t, (σ stands for saturation). Finally, the last symbol is a star notation " (*) " which means that a certain sequence is repeated zero or more times. This notation makes sense when a recursive rule is met. For instance, $(\sigma F_{t1}\ \sigma F_{t2})^*$ means that the sequence of firing of t1 and t2 is repeated up to saturation. The set of words built from the above symbols defines the PCN language noted L_{PCN}.. A more complete presentation of this language can be found in [24].

4.2 Functional architecture

The current prototype which implements the ideas described in this paper is an extension of the Sabrina relational DBMS [25, 26]. We first present the architecture of this system and then detail the alterations made to support the rule language with complex domains. The current architecture of the DBMS is composed of three layers of abstract machines, going from the end-users to the disk units:

(1) The interface machine comprises the external layer. It is responsible for the dialogue with the end-users and the parsing of the user requests into internal messages constituting an application protocol called the Data Manipulation Protocol (DMP). Several types of user interfaces can be offered.

(2) The assertional machine which constitutes the intermediate layer of the system performs the evaluation of relational tuple calculus assertions in terms of an extended relational algebra. This machine also includes integrity, a view mechanism and security control.

(3) The algebraic machine which is the most internal layer carries out the relational algebra operations as fast as possible. To supply this function, it manages the access path model based on predicate trees, uses a cache memory and implements efficient join and filtering algorithms. In addition, the algebraic machine performs the physical controls, that is concurrency and reliability controls.

Each machine is divided into *functional processors* which are implemented as software modules. The extension of this system did not change the global architecture but essentially changed the

organization of a machine and added new functional processors to a machine. Figure 5 represents the functional architecture of the system. First, the interface machine was extended to support the rule language as described in sections 2 and 3. The main task of the parsing processor is to perform some syntactic controls over the rules and then to compile them into a PCN form as shown in the previous sub-section.

The main changes brought to the system resulted in the assertional machine resulted in of A *PCN manager* was designed to efficiently retrieve the relevant rules associated with a query and to bring them in main memory under the form of a global query PCN. The evaluation of the query is then done by the query *PCN evaluator processor*. After a query optimization step, this processor produces a sequential execution plan written with the PCN language. Each instruction of this plan contains references to the PCN (contents of the transition, tuples contained in the places of the net). Then, the processor determines which transitions can be computed using a relational algebra program. A *relational algebra program* contains all the relational algebra operations plus functions, aggregates and "while ... do" control structures. The whole resulting execution plan is then treated by the optimizer processor [26]. The optimizer processor used in the present version of the prototype is svary close to a standard relational optimizer. Thus, iteration which arises in recursive rules is supported by the PCN evaluator processor. The optimizer processor evaluates the condition of each rule of the PCN and generates calls to the algebraic machine for evaluating all comparison predicates. When a rule is fired its action is evaluated and new calls to the algebraic machine can be generated. The only modification made to the algebraic machine was to incorporate the complex domain processor. The complex domain processor is called to evaluate the user-defined functions over complex domains contained in the comparison predicates or in the actions. This leads to (i) load the user-defined function into a LISP working memory and (ii) to evaluate the function.

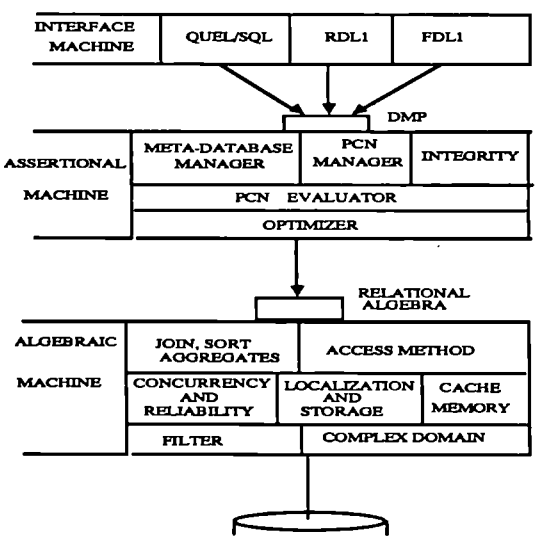


Figure 5: Functional architecture of the extended DBMS

4.3 The PCN evaluator processor

A query can be seen as a production rule represented by a single transition, several input places and a single output place, representing the result of the query. Thus, a query can be represented as the combination of two PCNs. The former represents all the rules needed to define the stable marking of the input-places of the query. The second net represents the query itself. The resulting PCN is called a *query PCN*. A query PCN poses the essential problem of choosing at each step, a transition firing occurrence among the transition's conflict set. We propose two basic techniques to obtain an efficient execution and to guarantee a determinist execution of a query.

The first one is to provide a meta-control which permits an ordering in the transition's firing which guarantees a determinist execution of a large class of rule programs: As mentioned before, rules are not given by the user in a predefined order. Thus, it is the responsibility of the system to decide in which order the rules have to be executed. We introduced a partial ordering over transition's firing occurrences which permits a computation by *stepwise saturation* of the transitions. This partial ordering is done through firing a meta control [15] which can be seen as an implementation of the concept of stratification [28] in a production rule language. For instance, on the net figuring the reduction module (figure 3) the rule t1 will always preceed the transitions t2 and t3 while transition t4 will be fired at last. However, in this example, our meta control is not able to order t2 and t3. If a particular ordering has to be chosen, it can be explicitly given by the user. This meta control has two important consequences :
(1) it allows an efficient computation of the PCN because the non recursive transitions are fired only once (by using a relational computation).
(2) it provides a determinist semantics of a large class of rule programs [20].
The result of applying this meta-control strategy is a sequential program of statements written in the PCN language presented in section 4.1.

The second technique is to exploit, whenever possible, the set-oriented processing capabilities of the relational DBMS for evaluating a single transition. As described in the previous section, the semantics of a PCN leads to what is usually called a "tuple-at-a-time" computation. In order to compute efficiently a PCN with a RDBMS, we wish to transform this "tuple at a time" computation into a set oriented one using a relational algebra program. This means that given a transition t, our objective is to transform the transition condition of t into a relational algebra expression which is computed using in one time all the tokens of the input places of t. In this case, we say that we perform a *relational firing* of the rule modeled by t. For instance all non recursive rules can be computed using a relational algebra program. However a relational firing of a transition is not always possible. More precisely, a relational computation of a rule does not necessarily return the same result as the execution mode induced by the semantics of a PCN.

Consider for instance the serial rule given in the module REDUCTION in section 2. It replaces two serial edges by one resulting edge. It is clear that a set oriented computation of the rule does not

produce the same results as a tuple-at-a-time computation. A counter example is presented in Figure 6. Let us consider three serial edges (A,B), (B,C) and (C,D). The set oriented computation produces two resulting edges (A,C) and (B,D), and the "tuple-at-a-time" computation produces one edge (A,D).

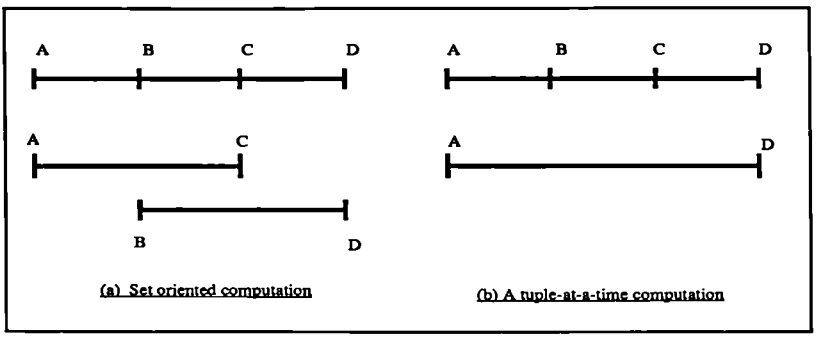


Figure 6 : serial rule computation.

For a recursive rule, we provided sufficient syntactic conditions to determine the relational computability of the rule. A more detailed discussion of this topic can be found in [15].

4.4. The complex domain processor

The evaluation of user-defined operators does not alter the logic behind the processing of standard database operators {+, -, *, ...}. It simply implies recognizing whether or not an operator is user-defined; if it is then the complex domain processor is called otherwise the standard processors are used. The processor involved in projection and restriction evaluates operators (user defined or specialized hardware) on a "one-tuple-at-a-time" basis and the processor involved in aggregate computation evaluates operators which handle sets of tuples. These processors are aware of the LISP interpreter and call the interpreter to calculate user-defined operators.

In our approach, the mapping of domain values to database attributes is made possible by representing hierarchical structures in text format with the use of parenthesis as is the case for any LISP structure. The passage of parameters from database values to the LISP processor is via the processors which handle projection, restriction and aggregates that, in the case of LISP functions, call the interpreter with the function name and function arguments to retrieve the function result. The simplified version of the procedure which handles a function calculation on a complex domain for projection or restriction is :

```
procedure complex_calculation (R: relation, C: calcul)
  for each tuple in R do
```

call interpreter (C, attribute values);

So, in accordance with the architecture presented in section 4.2, computations and aggregates are handled by the algebraic machine.

5. QUALITATIVE ANALYSIS

5.1 Functionalities and limitations

This paper present an integration approach to Rule data language with a DBMS. This integration is partially achieved by the compilation technique introduced in section 4. The new Petri Net based model used to represent a rule program plays a key role in the compilation process. A PCN is used to dynamically control the execution of a rule program. This is done through a particular language which models the flow of control in query processing strategies. In particular, the main known query optimization strategies for recursive rules can be captured by a PCN. Some query optimization techniques are presented in [20].

The integration of the rule language in a DBMS has several properties that appear as precise requirements of the potential applications studied in the project. An important point is the capability of expressing *dynamic updates* and sides effect as actions [2]. This means that based or derived relations can be updated at any time during the execution of the rule program. Furthermore updates are not restricted to the relations present in main memory, but can affect the all knowledge base. For example the action of a rule can update relations resident on disk. These updates can trigger the activation of new rules that were activated before. Thus, there is no separation between the deductive process and the updating one. They are intimately related. These features are required, for instance by the ITS (Intelligent Training system) application [2].

As for comparison, current logic programming languages used for deductive databases do not offer this functionality of dynamic updating. In Datalog languages, the update commands are separated from the inference process. In a PROLOG program, update actions are expressed as ASSERT or RETRACT predicates. However, the semantics of these actions is very cumbersome and since their implementation is dependant on the interpretor , no integration is possible with a DBMS.

On the contrary, the RDL1 language includes as a particular case, the operational semantics of Datalog languages. The operational semantics of the language provides a uniform and clean compromise between declarativity and procedurality. This makes writing and maintaining large rule programs easier. Indeed, the procedural aspects of RDL1 stand upon (i) the use of a kind of implicit stratified execution of a program which leads to an implicit ordering of the rules and (ii) the capability of specifying an explicit (user given) partial ordering of rules when ambiguous interpretations arise. Finally, RDL1 aims at being a full database programming language. All the rules are defined using modules. A module is the compilation unit of the language. The modules are hierarchically defined and allow a style of programming by stepwise refinement. The need for this style of programming has been clearly pointed out by an analysis of the applications our project.

The implementation of complex domains provides for an extended relational model where complex

domains are integrated at all DBMS levels: (i) the user can access FDL1 from within the assertional language (there is no external programming environment), (ii) new domains and operators are registered with the system and their respective code is stored within the system in a function base (no external files are needed), (iii) complex domains are organized in a "IS A" hierarchy for operator inheritance among domain types (iv) new domains and operators can be defined using existing ones, (v) a complex domain clustering strategy can ensure efficient access for frequently applied functions, (vi) an object oriented "on the top" interface can be implemented using the same version of the LISP language as the "on the side" extension, both implementations can share the same function base for user programs.
Several query optimization techniques have been developed in order to compute efficiently a PCN [20]. In particular, specialized techniques have been specified to deal with recursive rules. The model is general enough to include in a uniform way the new data types defined through the FDL1 language.

5.2. Performance issues.

A first implementation of the rule language and its execution model has shown the ability of our approach to specify complex applications such as the ones studied in the Esprit project ISIDE. Also, it pointed out some research perspectives for improving both the usability and the performances of the system. For the RDL1 language, performances presents two main problems : (i) to provide techniques for a global optimization of the extended relational algebra programs which are produced by the analysis of the PCN; for instance, these techniques should take into account particular access methods and an intelligent managing of temporary results in main memory (ii) to design specialized fix-point operators for dealing with recursive rules [29]. These operators are based on the traversal of join graphs. The graphs are implemented with efficient data-structures in main memory and/or disks. This technique can be extended with weight computations, allowing the system to support efficiently practical examples of recursion. Such operators would replace the iterative calls to the algebraic machine done by the PCN evaluator.
The limitations of this implementation of complex domains stems from the fact that it is based on a loose coupling strategy. In this type of strategy, before any computation can be done, values have to be transformed from their text representation into the interpreter's working memory. To do so implies the managing of a symbol dictionary to ensure symbol unicity. It logically follows that this limitation is particularly penalizing when processing large structures. The size is also limited to the size of working memory. This limitation will be overcome with a tight coupling strategy with the algebraic machine. A new algebraic machine is being developped to support a rich variety of structures which the interpreter will directly manipulate. Therefore, the distinction between LISP working memory and database storage structures will disappear and large structures will be processed efficiently. Compared with an interpreted strategy, a compiled strategy suffers from the problem of linking the user-code to the DBMS program and protecting the DBMS program from errors arising in the user code which could cause it to terminate abnormally. But, the advantage of a compiled strategy over an interpreted one is the gain in performance when processing large

numbers of tuples.
To insure performances in the retrieval of complex values which involve operators, a clustering method based on predicate trees [30], [31] has been extended to include complex domains with operators [32]. An efficient clustering method should allow clustering tuples verifying an identical function result. On one hand, the retrieval of rectangles such that SIDES = (3 4) can be optimized by a clustering on the value of the attribute. On the other hand, if the selection is on surface value begin equal to 12 (where SIDES.SURFACE = 12), a clustering organization is efficient only if it is based on the SURFACE function. In such a case where standard clustering methods require processing the entire relation, our approach allows searching only those blocks possibly containing the desired result by using functional clustering based on surface values.The following assertional command clusters tuples at a first level with a hashing function applied to rectangle numbers and clusters tuples at a second level with the same hashing function but based on the result of the SURFACE function applied to the SIDES attribute.

CLUSTER RECTANGLES ACCORDING TO
(NO ML04)
(SURFACE (SIDES) ML04)

Performances can also be insured by the intelligent filtering of queries. LISP structures are stored in attribute values as text where the hierarchical structure is represented with parenthesis as is the case of any LISP structure represented in text format. Consider the following example using the definition of a dimension which comprises the domain of rectangles built from two numbers. The following function defines the structure of a rectangle:

(DEFSTRUCT DIMENSIONS LENGTH:integer WIDTH:integer)

In this example, the two values which comprise a rectangle take their values from the integer domain. Although, the individual domains can be any other complex domain.
Before attempting evaluation, the correct LISP function must be selected from the function base and loaded into LISP working memory. This work is minimal inlight of the fact that user operators defined in the form of LISP functions are interpreted. Although, the slower interpreted processing is penalizing when processing larger quantities of data. Before processing the complex value, the interpreter must transfer it in LISP working memory which implies creating memory addresses and managing the symbol dictionary. Even in an optimal environment, a weak coupling strategy implies that this be done for every tuple value.

We propose a new strategy for processing extended queries which involve only the schema of complex domain structures. Such queries can be resolved without the LISP interpreter through the use of a query modification strategy involving text operators. We can thus define two classes of extended queries, those which can be resolved without the interpreter and those which require calls to the interpretation process. Namely, the former class of extended queries are those which involve

only the domain structures. The intelligent filtering operation uses text operators which isolate the desired value from within the complex structure.

6. CONCLUSION

In this paper, we proposed an extension of a powerful rule based language to support complex domain in a DBMS. A rule consists of a conditional part which is a tuple relational calculus expression and of an action part which is a sequence of insertions and deletions of tuples in the database relations. RDL1 offers more generality and more computational power than the Datalog languages and provides a good compromise between declarativity and procedurality. A general compilation technique is used to transform the rules into a new execution model based on Predicate Transition Network. This model serves as a basis for an integration between the rule language and the DBMS. Several query optimization techniques are supported by the model. In particular recursive queries can be handle efficiently.

The implementation strategy for complex domains is one of a loose coupling between the RDBMS and a special purpose LISP language interpreter. Complex attribute values are stored as text strings where the hierarchical structures are represented using parenthesis. In this loose coupling strategy, performance penalties are incurred when processing large structures which need to be loaded into LISP working memory for processing. Present work involves building a new algebraic machine with extended representation capabilities for a tight coupling strategy. Large objects will thus be processed as easily as small ones since the interpreter will operate directly on the structures of the algebraic machine.

The object oriented paradigm has gained much interest in the area of persistent memory support. The problem raised is the creation of a new DBMS model based on the object oriented model. Although, research has also been centered on the creation of an object oriented interface over an existing DBMS model. As presented earlier in this paper, the SABRINA DBMS has been extended to include complex domains by an "on the side" extension using a specialized object oriented LISP language processor. An "on the top" interface will be built over the DBMS using a compatible LISP language processor which will use the same facilities as the "on the side" extension. Hence, all functions will be stored in the function base. The mapping of objects represented by classes can be directly achieved through relations extended to include complex domains.

A prototype of each system has been developed at INRIA and at the University of Paris VI and they are both operational on a DPS8 computer running under the MULTICS operating system. Present work involves the integration of the two.

REFERENCES:

[1] Deliverable report on task 1 on Work package 1, Esprit Project 1133 april 87.
[2] Deliverable report on task 3 on Work package 1, Esprit Project 1133 to appear October 87.
[3] J. Ong, et al.: *"Implementation of Data Abstraction in the Relational Data Base System INGRES"*, SIGMOD, rec 14, 1,pp1-14, 1984
[4] M. Stonebraker, et al. : *"Application of Abstract Data Types and Abstract Indicies to CAD Data Bases"*, Proc. of Engineering Design Applications of

ACM-IEEE Database Week, San Jose, Ca., May 1983
[5] M. Stonebraker : *"Inclusion of New Types in Relational Database Systems"*, ACM-IEEE,1986, pp 262-296
[6] S. Tsur, C. Zaniolo : *" An Implementation of GEM / supporting a semantics data model on a relational back-end"*, Proc. ACM-SIGMOD Conference on Management of DATA, 1984
[7] C. Zaniolo : *"The Data Base Language GEM"*, Proc. ACM-SIGMOD Conference of Management of DATA, 1983
[8] C. Zaniolo, et al. : *"Object Oriented Database Systems and Knowledge Systems"*, MCC Technical Report No. DB03885
[9] A. Goldberg, D. Robson : *"Smalltalk-80 : The language and its implementation"*, Ed. Addison-Wesley, 1983
[10] P. Schwartz, et al. : *"Extensibility in the Starburst Database System"*, Proc. of the International Workshop on Object-Oriented Database systems, California, Sept86, pp. 85-93
[11] S. Osborn, T. Heaven : *"The Design of a Relational Database System with Abstract Data Types for Domains"*, ACM Transactions of Database Systems, Vol11, No.3, Sept.86, pp 357-373
[12] J. Chailloux, et al. : *"Le_Lisp Version 15.2 Reference Manual"*, Ed. INRIA, Nov. 86, 3rd Edition
[13] G. Kiernan, R. LeMaoult, F. Pasquer : *"The Support of Complex Domains in a Relational Database System Using an Integrated LISP language processor"*, 3ième Journées Bases de Données Avancées, Port Camargue, Mai 1987
[14] E. Simon : *"RDLI : A Production Rules Language for Deductive Databases."* INRIA Internal Report, April 1987.
[15] C. de Maindreville, E. Simon : *"Deciding if a production rule is relational computable" Journées FIRTECH , Ecole Normale Superieure, April 1987.*
[16] E.F. Codd : *" A Data Base Sublanguage founded on the relational calculus."* Proc of ACM SIGFIDET 71.
[17] J.M Nicolas, R. Demolombe : *"On the stability of relational queries "*, Proc of Int Workshop Logical bases for Databases, CERT, Toulouse, 1982.
[18] L. Brownston, R. Farrell, E. Kant, N. Martin :*"Programming Expert Systems in OPS5 : An Introduction to Rule-Based Programming"*. Ed. Addison-Wesley.
[19] F. Bancilhon, R. Ramakrishnan :*"An Amateur's Introduction to Recursive Query Processing Strategies."* Proc of ACM-SIGMOD , Washington D.C., may 1986.
[20] C. de Maindreville, E. Simon :*"A Predicate Transition Net for Evaluating Queries against Rules in a DBMS."* INRIA Research Report, N° 604, Feb. 1987.
[21] J.L. Peterson : *"Petri Net Theory and the Modelling of systems."* Prentice-Hall.
[22] H. J. Genrich : *"Predicate / Transition Nets "* , in Advances in Petri Nets' 86. Springer Verlag, 1987.
[23] A. Giordana, L. Saitta : *"Modeling Production Rules by Means of Predicate Transition Networks."* Inform. Sciences Journal, North Holland Ed. Vol.35, N°1.
[24] C. de Maindreville, E. Simon : *" A production rule based approach to deductive databases"* Submitted to publication.
[25] G. Gardarin et al. : "SABRE : *A relational database system for a multi-processor system*", Advanced Database Machine Architecture, Book, D. Hsiao Ed.,Prentice Hall, 1983.
[26] G. Gardarin, C. de Maindreville, D. Mermet, E. Simon : *"Extending a Relational DBMS towards a KBMS : A First Approach ."* Proc. Workshop on Knowledge Base Management Systems, Ed. J. Schmidt, C. Thanos, Crete, june 1985, to appear Springer Verlag Book.
[27] G. Gardarin, E. Simon, S. Abiteboul, M. Scholl : *"Towards DBMS's for supporting new applications"*, 12th VLDB Int. Conf., Kyoto, 1986.
[28] K.R Apt, H.Blair, A.Walker : *"Towards a theory of declarative knowledge"* IBM Research Report RC 11681, april 1986.
[29] G. Gardarin, P. Pucheral; :" *Optimization of recursive queries using Graph Traversal"* Internal Research Report, INRIA.
[30] G.Gardarin, P.Valduriez, P.Viemont : " *Predicate trees : a way for optimizing relational queries"* proc of the computer Enginneering Conference, IEEE, Los Angeles, April 84.
[31] P.Valduriez, Y.Viemont : *"A multikey hashing scheme using predicate trees"* ACM SIGMOD Int Conf on Management of Data, Boston, June 84.
[32] J.P. Cheiney, G. Kiernan : *"A Functional Clustering Method for Optimal access to Complex Domains in a Relational DBMS"*, submitted for publication

Project No. 1098

How to Build Knowledge-Based Systems: Techniques, Tools and Case Studies

S A Hayward
STC Technology Limited
U.K.

Abstract

Increasing numbers of software developers wishing to build knowledge based systems are realising the need for well-founded techniques to guide and direct the development process. The results and experience from Esprit Project 1098 (A Methodology for the Development of Knowledge-Based Systems) are now making the transition from ideas and concepts to practical methods in use for commercial systems development. The methods are supported by a computer based documentation system and a detailed Handbook.

The project has also produced a programming system as a basis for its own tools which is now being used by a number of other Esprit Projects and is being sold to other research groups. This system adds the power of object-oriented programming to symbolic processing languages (such as Prolog and Lisp) and is particularly suited for the development of systems with sophisticated graphic based interfaces which are nevertheless simple to use and easy to program.

This paper summarises the current status of the methodology and illustrates some of the projects on which it has been used. In particular the techniques for the analysis phase of a development are now reasonably mature and should provide the basis for construction of a wide range of types of knowledge-based systems. They are now also being used by personnel outside the research team.

These results are primarily concerned with the "internals" of KBS development. However as we move from the early "technology push" phase, where a KBS may be considered desirable in itself, to an "application pull" phase, where the primary concern is the solution of the problem irrespective of the technology used, then it becomes increasingly important to understand the relationship between KBS technology and other software technologies. In this context the relationship is not primarily in terms of technical comparisons but in terms of how an organisation decides to use one or more and how such "external" factors impinge on the development process. The methodological view, as evolved in P1098, is well able to accommodate such issues. We therefore believe that as well as being technically sound it is useful and robust in the context of realistic organisation constraints.

1. THE SOFTWARE DEVELOPMENT PROCESS

We start by examining software production as another instance of a commodity producing process. At the most general level, if one considers the whole range of endeavours which are intended to produce man-made artifacts they can be seen as a spectrum ranging from craft to industrial production. Crafting is an essentially unstructured process in which progress is made towards a goal (often not specified in any publically inspectable form). The process is not readily partitioned or may consist of many small steps which cannot be differentiated but make a cumulative contribution to achieving the goal. Extreme examples of such activities are sculpture and pottery-making. At the other end of the spectrum is fully industrialised production, for example of mass-produced consumer goods. Such production is highly structured, with each component of the process rigourously defined. This leads to benefits which have been much analysed by economists, including efficiency in materials procurement, quality control, division of responsibilities between work-groups, etc. One key characteristic of this end of the spectrum is a clear specification of the cycle of production i.e. production is defined as proceeding in a number of stages, each discrete and well defined, with well-defined relationships to preceding and succeeding stages.

The production of software generally lies somewhere between these two extremes. The use of the metaphor "software factory" clearly indicates an aspiration towards one end of the spectrum. However the traditional view of AI programming tends more to the other end of the spectrum. It is a fundamental presupposition of our research that commercial KBS development can and should be viewed as towards the industrialisation end of the spectrum. Thus a description of the development (production) process for such systems in terms of a lifecycle is appropriate. It also provides a descriptive framework within which many critical global issues can be tackled - notably the inter-relationship of activities within the development process and the control of iteration within the process.

At the top level the KBS development lifecycle (see Figure 2) is very similar to any conventional software lifecycle with the usual phases of analysis, design, implementation and so on. It is at the next level of detail that more significant issues emerge. A key insight with regard to the analysis phases the distinction between *internal* and *external* views; the former covering the "knowledge acquisition" activities (knowledge analysis, analysis of expert and user tasks, and construction of a conceptual model) and the latter the "requirements analysis" activities (analysis of the current situation, analysis of organisational objectives and constraints, determination of functional requirements). There is an interplay between these two views which is focused by a scoping activity (i.e. the extent of the project/prospective solution) and feasibility estimation. (This is summarised in Figure 1 and presented more formally in Figure 3.

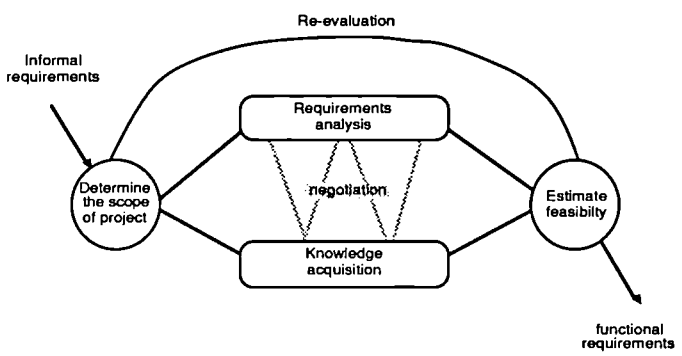


Figure 1 Scoping / Feasibility cycle

In retrospect we can see that difficulties in a number of our analysis projects have been caused by the inadequate separation of these two views. Equally the clarification of the objectives of each of these activities and the output associated with them has enabled project planning and division of tasks in more recent analyses to be greatly improved. Thus we appear in this respect to to be achieving the objectives of using a lifecycle perspective.

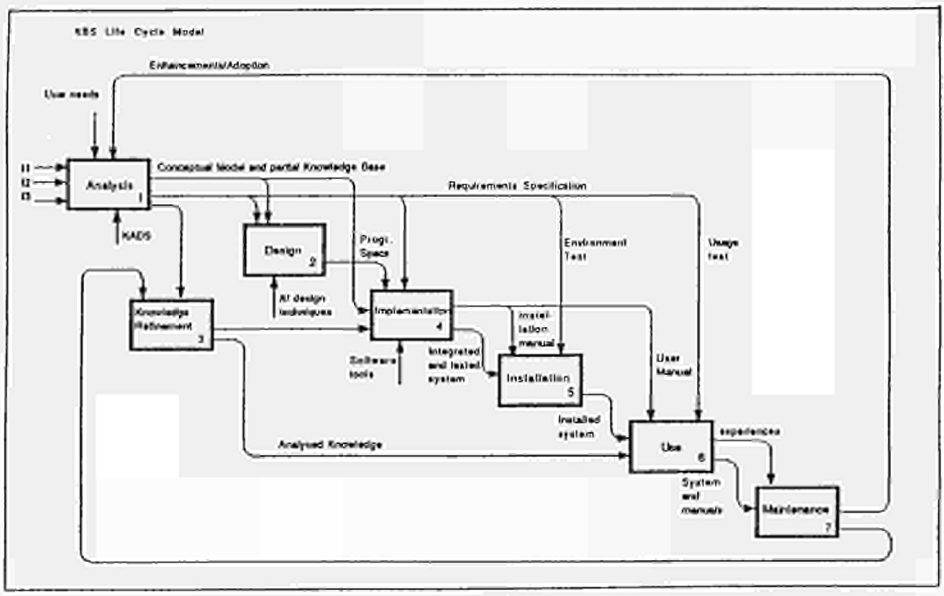


Figure 2 Top level Life Cycle Model

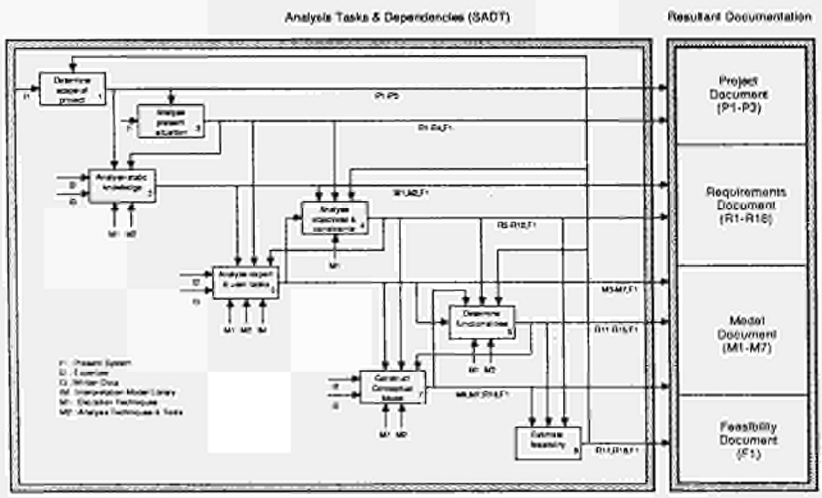


Figure 3 Model of Analysis Phase

The definition of a scoping activity and its relation to these internal and external views also helps to clarify a wider isssue, namely, how does the decision to base a development on KBS technology arise in the first place. It may of course be an initial constraint, a classic case of the early "technology push" phase in the evolution of a technology. However if we look at the database world and the world of decision support systems it is clear that they also have evolving technologies which draw on many ideas similar to those current in the KBS field. The relationship between these technologies is illustrated in Figure 4 which shows them as based on similar core concepts (which are converging) and impacting through different levels of business constraints. Each technology currently has its own *internal* view and as such must be assessed independently for its capability to contribute to satisfying a need. However as the core converges and as corporate strategy increasingly demands solutions independent of technology constraints there must be a convergence between the technologies overall. This evolution can be understood in terms of a shift from multiple internal views to an increasingly sophisticated scoping operation (cf Figure 1) which acts in a nested or recursive way from the most general aspects of corporate strategy down to determining appropriate responses to specific operational problems.

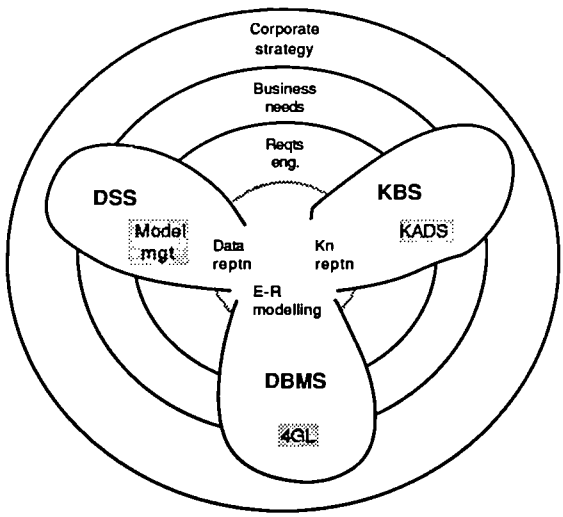


Figure 4 Software technologies in a business environment

Another major issue which can be clarified within the life cycle perspective is the function of prototyping in system development. One view of prototyping tends to view it as an alternative to other activities, particularly analysis and modelling activities. In this view prototyping must be presented as an additional or alternative phase in a methodology. We sugggest that this is an incorrect understanding of the function of prototyping, which should rather be considered a technique which can suppport many activities. In SADT terms prototyping is a *mechanism* which supports activities, not an activity in itself. The question as to the function of prototyping can now be seen more clearly. To answer it we must analyse where prototyping may be used in the lifecycle and for what purpose. This will lead to a categorisation of prototyping and the decision as to whether to employ the technique in particular instances will be made on a case by case basis.

There is already a substantial amount of work on prototyping within the Data Processing community, arising from the Research Programme 'Specification and Development of Software Systems', undertaken by the National Computing Centre (UK) and the Gesellschaft fuer Mathematik und Datenverarbeitung (W. Germany), and subsequent conferences and workshops (NCC, 1985). We believe that much of this is relevant to the KBS lifecycle within the framework described above.

Five categories of "generic issues" are identified to which prototyping may be relevant:

- system requirements
- design of a solution
- resources
- effects on the organisation
- changes in the outside world

There are also five main types of prototyping distinguished:

- exploratory
- experimental
- performance (sometimes known as 'synthetic')
- organisational
- evolutionary

Evolutionary prototyping corresponds to the paradigm of an experimental methodology as we have characterised it above, and we note the comment made: "What might be called 'incremental evolution' could well be a risky business". The use of exploratory prototyping is considered in the context of Requirements Analysis in Barthelemy et al (1987). Experimental and performance prototyping are more likely to be appropriate in the design phase, with the exception of experimental prototyping as a mechanism for capturing system-user interface requirements (which is recognised for all types of software development). Organisational prototyping lies outside the main focus of P1098, given its concentration on technical issues rather than social and organisational ones.

In addition to categorising the function of prototyping we must analyse its impact on the development process. Dearnley and Mayhew (1983) have characterised this in a lifecycle consisting of two interacting cycles, emphasising the difference from a more traditional linear view of the lifecycle. A slightly modified version of this model as a basis for consideration in P1098 is shown in Figure 5. The KBS lifecycle requires more refinement, particularly in the design phase, before we can produce a KBS equivalent of this view but the work to date (notably the notion of nested levels of requirements analysis) seems entirely consistent with it. We would expect in any event that the issues addressed here are relevant to all software lifecycles and that the KBS lifecycle would be a variant rather than dramatically different.

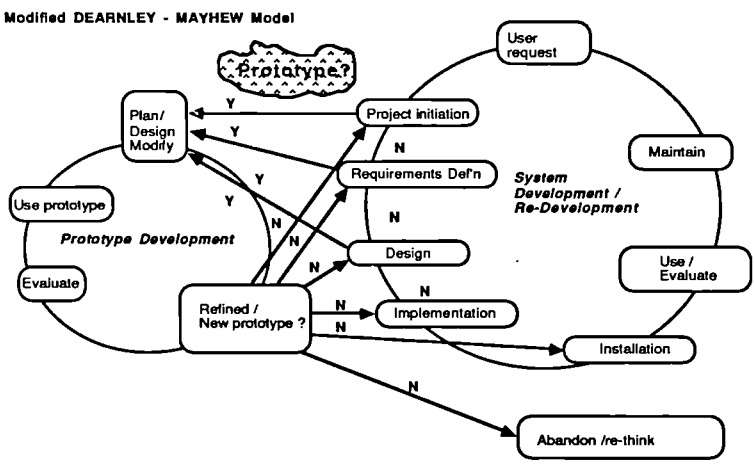


Figure 5 The interaction of Prototyping with the Software Development Lifecycle

2. MODEL DRIVEN KNOWLEDGE ACQUISITION

Knowledge acquisition is the major activity driving the *internal* stream of the KBS analysis phase. In this context a major problem facing a knowledge engineer when constructing a KBS is organising and understanding his data. This process, which we term *interpretation*, is essentially one of the knowledge engineer building a model or models into which he tries to fit this data. This modelling process has generally been left implicit in previous descriptions of the system building process. This is unsatisfactory because it fails to make the model building and testing open to inspection and critique, and it also does not allow the provision of support or constraints for the process. Another particular difficulty is that typically the models used by AI programmers building knowledge based systems will be implementation oriented, i.e. they will think in terms of the data structures and procedures they expect to see in the implemented system. The problem with this is that it is in general very difficult to map the data about the expertise and the domain onto such constructs: the descriptive vocabulary is too distant from the data. This we suggest leads to much of the perceived difficulty of knowledge acquisition. Our objective in this area is to make available an "appropriate" modelling language and make the model building process explicit, and to a degree provide normative support for it.

One specific characteristic of our modelling language is that it allows us to represent problem-solving activities at a fairly high level of abstraction/generality. This means that we can also provide models which describe the prototypical character of classes of problem-solving activity. The knowledge engineer can use such models as a starting point for data analysis by seeing to what extent his data fits an existing model or composite of models. Thus the knowledge acquisition process becomes not only explicitly one of model building but one of model refinement rather than model creation. This we believe is a less demanding task and thereby enables more effective knowledge acquisition by less experienced people.

The modelling language

The modelling language derived in P1098 incorporates four layers of description, each containing different types of knowledge. We distinguish between static knowledge describing concepts and relations, knowledge of different types of inferences that can be made, knowledge representing elementary tasks, and strategic knowledge. Each of these categories of knowledge is described at a separate level, reflecting the different ways in which the knowledge can be viewed and used.

The first layer contains the static knowledge of the domain: domain concepts, relations and complex structures, such as models of processes or devices. The second layer is the inference layer. In this layer we describe what inferences can be made on the basis of the knowledge in the first layer. Two types of entities are represented at the inference layer: meta-classes and knowledge sources. Meta-classes describe the role that domain concepts can play in a reasoning process. For example, a domain concept like infection can play the role of a finding in a consultation process, but it may also play the role of an hypothesis. Knowledge sources describe what types of inferences can be made on the basis of the relations in the domain layer. Examples are specialisation and generalisation knowledge sources, which both make use of a subsumption relation in the domain layer. These two layers provide what we may think of as a "theory" of the domain. They define what can be known and infered but say nothing about how such knowledge is actually applied to reason towards desired conclusions.

The third layer is the task layer which now begins to define how the knowledge expressed in the two previous layers is used in a problem-solving context. At this level the basic objects are goals and tasks. Tasks are ways in which knowledge sources can be combined to achieve a particular goal. The fourth layer is the strategic layer in which knowledge resides which allows a system to make plans - i.e. create a task structure -, control and monitor the execution of tasks, diagnose when something goes wrong and find repairs for impasses. These four layers are schematically presented in Figure 6.

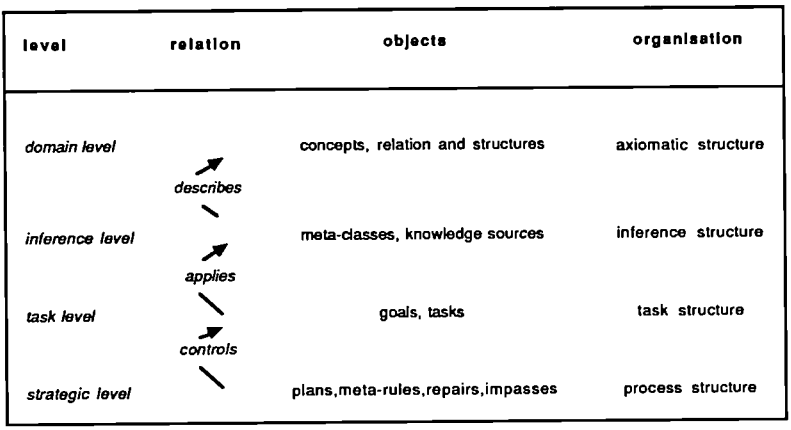


Figure 6 Layers of the four level model

Creating models

We distinguish two sets of models that are used in the KADS methodology, although both are described using the four layer notation outline above:

conceptual models

The conceptual model is a full model of the expertise that is to be implemented in a future knowledge-based system. It contains a full description of knowledge at all four levels described above. The conceptual model is a major output from the analysis phase and a primary input for the design of the system.

interpretation models

An interpretation model is a description of the knowledge required to perform a particular class of tasks, given in terms of the inference, task and strategic levels. The interpretation model generally does not specify the nature of the domain level knowledge.

Interpretation models can be further divided into models of so called 'generic tasks' and 'real life tasks'. An interpretation model can be viewed as a conceptual model abstracted from its domain specific features, i.e. its domain layer, and its specific references from the inference structure to this domain layer. In general a model that is obtained in this way is called a 'real life model'. However often real life expertise may be viewed as the combination of various elementary tasks, which we term 'generic tasks'. Thus we distinguish:

Generic Models

These are interpretation models for elementary problem solving tasks. An elementary problem solving task is a task that can stand 'on its own': i.e. has as its input a problem and produces a solution. This solution may be used in other tasks, but it is the answer to some original problem.

Real Life Models

Real life models are composites of generic models. Although real life models are abstracted from a specific domain, there may still be some higher level domain dependencies left in the model, which may constrain the scope of applicability of real life models. Real life models may assume certain similar structures across domains, or even similarities in the functionality of the expertise that has lead to a particular composition of tasks.

Generic models are thus the essential ingredients in the use of interpretation models in KADS. We assume that the number of generic tasks -or, of protypical generic tasks- is a manageable set: preferably described as a taxonomy. Considerable effort has therefore been spent on developing generic models, and the taxonomy of available models is shown in Figure 7.

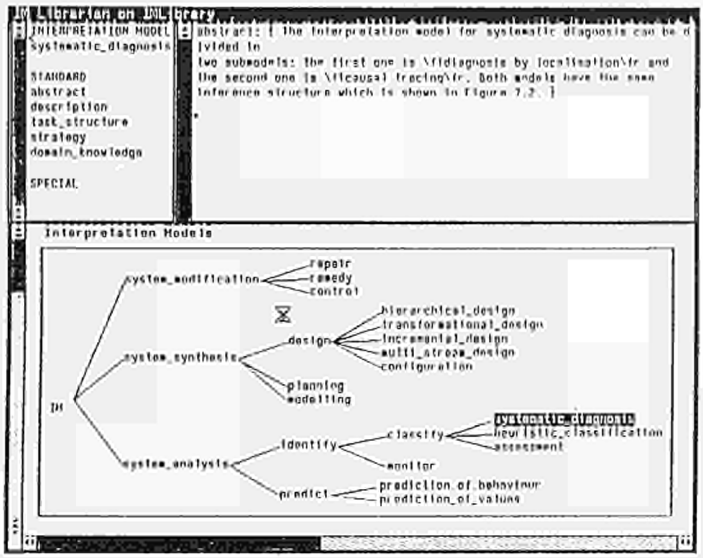


Figure 7 The current set of Interpretation Models

3. A BASIS FOR KBS DESIGN

Accepting the possibility of creating models of a domain and associated expertise as a basis for KBS development (Breuker et al, 1987; Johnson and Gruber, 1986; Clancey, 1986), the obvious question arises as to what relationship exists between the model resulting from analysis and the eventual systems as implemented. In a methodological context this is a particularly hard question because the process of developing the system given the analysis must be defined (and useable) if the methodology is to be of value. Experience in building systems starting with the KADS analysis (Breuker et al, op cit) suggests that given a better structured analysis the subsequent stages of development will proceed more easily, albeit still on an intuitive basis (Wielemaker 1987, Hayball 1986). This would be expected if the developer is in fact using internally generated models of the system to guide his activity (cf Littman; 1986). A structured analysis would then provide better support to develop this intuition.

Our methodological objectives thus require that we make the modelling basis of "design" explicit and that this modelling is characterised in terms of an appropriate level. By design we mean the decision process which takes a system requirement, given as a combination of internal and external views (see Barthelemy et al 1987) and determines what structural characteristics are necessary in the artifact that will meet this requirement. This is a level of decision making prior to actual writing of code but covers issues such as what implementation vehicle (e.g. Expert System shell) is suitable for this application, or how should components of a system be divided between the paradigms available in a multiple facility toolkit such as KEE, ART or KNOWLEDGE-CRAFT. More directly in relation to a KADS four-layer model, one has to decide how the levels of a specific model should be mapped onto an implementation vehicle. This can be seen as another instance of the "where do I start" question which is central to development of a methodology. The fundamental requirements for a design methodology are therefore a set of categories in which to describe the design options and a characterisation of design strategies or decision paths which

permit selection between these categories on the basis of inputs from the prior development process. These are currently being investigated.

4. THE METHODOLOGY IN USE

The methodology has now been used for a variety of development projects both within the P1098 research programme and independent of it. This is clearly important for two reasons: the methodology must be useful for real commercial applications and it must be useable others than those originally responsible for developing it. A number of the uses of the methodology have been for initial studies to assess the suitability of particular problem domains for the application of KBS technology, which probably reflects on the current state of evolution of the technology as much as the methodology itself. the following list shows the projects outside the scope of P1098 where the results have so far been applied or are currently in progress:

Bank loan assessment (study)
Social security assessment (study)
Credit guarentee assessment (joint project with an additional commercial partner who was responsible for the implementation of the system)
Research portfolio selection (implemented)
Oceanographic acoustic modelling (taken to demonstrator stage)
Information Retrieval assistance (study)
Nationality Law advisor (study)
Financial Planning
Machine tool fault diagnosis (study)
Computer maintenance
Manufacturing scheduling

Within the project two experimental systems have now been completed: a statistical advisor which is now being field trialed with students and a wide area network management advisor which has been demonstrated successfully to clients. A system to advise operators of a process involved in the manufacture of printed circuit boards is near completion and is expected to be operational in the second half of this year. Work is now also in progress on a system to assist designers of moulds for plastic piece parts and another to advise on the design of fluid mixing systems in the petrochemical industry.

This variety of applications and the fact that systems are now being developed for live operational use is for us one of the main indications that the results of P1098 are indeed proving valuable in aiding the exploitation of KBS technology, thus fulfilling one of the primary objectives of the Project, which was to assist the wider disemination of the technology in commercial practise.

5. SUPPORT TOOLS

The current set of support tools for the methodology, known as the KADS Power Tools (KPT), assist the application developer in a "bottom-up" way. In other words they provide passive support, allowing the recording of relevant data in various forms as analysis proceeds. We envisage that the normative guidance in applying the methodology, which is currently provided in the form of a Handbook, should also be incorporated in the support tools. This would make advice and guidance available in an interactive form, appropriate to the particular development in progress. However this is for the future. We will describe here the tools which exist at present.

The KADS Power Tools are structured around three editors which operate on different types of data, corresponding to three levels of analysis in the methodology: verbal, conceptual and epistemological. This is illustrated in Figure 8. The protocol editor allows processing of continuous text, which would normally be transcripts of interviews. The concept editor allows the definition of concepts, relations and hierarchies; corresponding broadly to the domain layer of the four-level model. The conceptual model editor handles the upper layers of the model, particularly the diagrammatic form used for the inference structure. This editor is closely related to the Interpretation Model Librarian which allows models to be stored, browsed, and selected as a basis for modification to produce a Conceptual Model for the application domain.

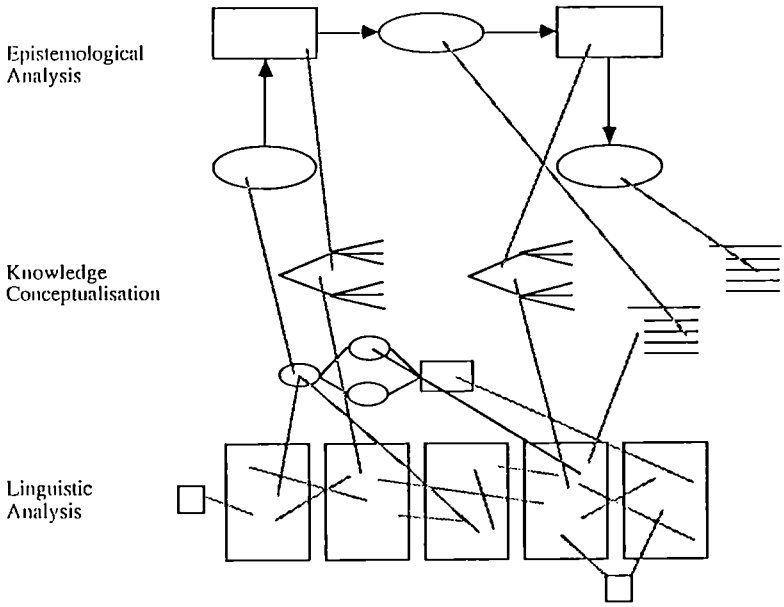


Figure 8 KADS Tools hierarchy

Protocol Editor

PED, the KADS Power Tools Protocol Editor, supports the linguistic or knowledge identification level in knowledge acquisition. PED is basically a dedicated hypertext system. Hypertext (Nelson, 1980) is based on the idea that information should be accessible in a non-sequential way, for instance someone reading a paper should be able to select a word and see it's definition, follow references or read other people's comments. Some hypertext systems have become widely available recently (Guide on the Apple Macintosh for instance), but there still is little agreement on what a general purpose hypertext system should provide and, perhaps more important, how it should be provided.

In PED, hypertext provides a solution to the problem of dealing with related written materials. Every hypertext system defines a set of link types between text. Nelson, for instance, defines jump-, quote-, note- and equi-links, which corresponds roughly with the way people want to read or write documents. The situation is different for protocol analysis. The knowledge engineer does not need to write a document, s/he must study the protocols and identify knowledge for later use plus topics and questions for further elicitation sessions. PED supports this analysis by allowing sections of a protocol to be linked together, or to concepts, notes, etc.

All link-types in PED are defined on *fragments*. A fragment is a contiguous user specified part of a text. Obviously, fragments may have any length, they could refer to a word, sentence, paragraph or even the entire protocol.

The following link-types are available in PED:

An *annotation* is similar to a footnote in a document. Annotations are used to make observations about a fragment, for instance to point out lack of knowledge or to describe the KEs understanding what the expert says. It is up to the knowledge engineer how s/he annotates a protocol.

A *group* is a named collection of fragments. The relation between the fragments may be superficial ("about-user"), about a domain concept ("micro-processor") or some high-level abstraction ("inferences"). The important point is that grouping allows the knowledge engineer to create collections of fragments which s/he thinks are related. Syntactic approaches, e.g. keyword matching, are necessarily more restrictive. Any fragment may be a member of any number of groups.

A *link* is a named cross-reference between two fragments. Named links provide a general solution to the hypertext problem of how many link-types should be provided. In PED the answer is one, but the knowledge engineer is able to name that link. Named links allow the knowledge engineer to create "contrast", "detail" and "continues" etc. links between fragments, thereby once again offering a facility to impose any desired semantics on the relationship between fragments.

A *concept-link* is a link between a fragment and a concept. The name of the concept-link can be used to describe attributes or give a definition. These links provide the primary means for connecting linguistic analysis in PED with conceptual analysis in the concept editor.

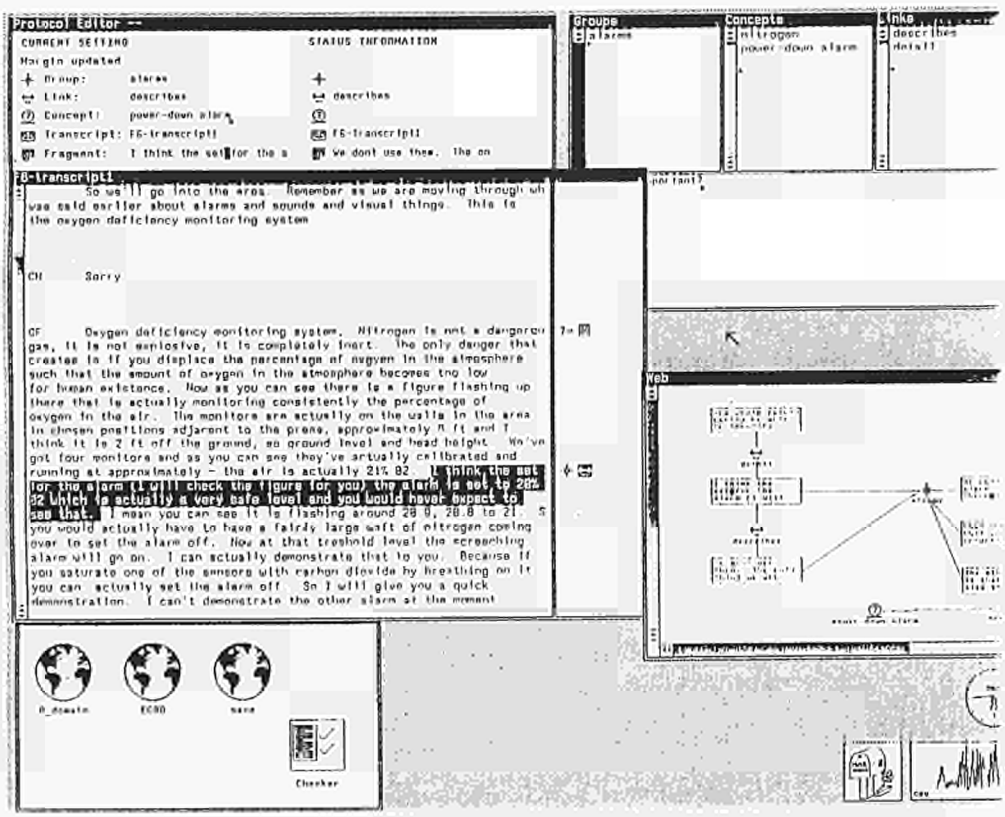


Figure 9 Protocol Editor

The PED user interface has an interesting feature which hopefully solves some of the potential user interface problems associated with hypertext systems. A protocol is displayed in a protocol window (large window in figure 9). A protocol window consists of a text part (on the left) and a margin (on the right). The margin contains a marker -shown as a small icon- for each link in the visible part of the protocol. The text part and the margin scroll simultaneously, the markers therefore always point to fragments in the text shown. The user interface makes clear that there are links in a given portion of the protocol and also identifies what type of link they are, as each link-type has a predefined marker icon. Analysing protocols in PED is like writing notes in the margin as the knowledge engineer might do on paper.

Finally PED will print arbitrary selections from the protocols. The knowledge engineer specifies a search criterion, for instance all links to a particular concept or all members of a group, and PED then collects, formats and prints the fragments found. Textual annotations turn into footnotes and the beginning and end of fragments are indicated. These print-outs can be used to communicate with the expert ("this is what I know about transistor-2"), for further study and for documentation purposes.

Concept Editor

The Kads Power Tools Concept Editor supports the knowledge conceptualisation level in knowledge acquisition. At this level knowledge is organised as concepts, relations and structures of concepts over a certain relation. The knowledge engineer needs a tool which allows incremental definition of concepts and the relations they have. As the knowledge engineer, at this stage, is still trying to understand what knowledge is important in the domain, the tool should be flexible and preferably not complain about possible inconsistencies other than gross syntactic errors, such as that a concept can not -indirectly- be a sub-concept of itself.

The Concept Editor knows that concepts may have attributes, and that these attributes may have a value. It also knows that concepts may be related to other concepts. Precisely what attributes and relations are used is highly dependent on the domain. It is up to knowledge engineer which attributes are important for the domain at hand, and there is no obligation to completely define these concepts.

The creation of taxonomies is a very basic form of conceptual structuring and thus the creation of class hierarchies (with inheritance) is provided as a primitive facility in the Concept Editor. However at the conceptualisation level other relations may also be important, e.g. consists-of, causal or temporal relations. Each hierarchy only shows the concepts over a given relation from a given root-concept, implying that a single concept can be in more than one hierarchy and that more than one disconnected hierarchy over the same relation may exist. The Concept Editor has implicit support for building a glossary. Each concept has four standard attributes (description,translation, source reference, synonyms) which are initially empty and can be filled by the knowledge engineer.

Documentation is considered important in all (KBS) methodologies, to quote Freiling et al. (1985; p. 158):

> All too often, knowledge engineering projects become a black hole, and managers have difficulty perceiving signs of progress. With a clear sense of stages and documentation which can be delivered at each milestone, it becomes possible to say "We've completed the knowledge organisation phase and we're now defining the representation," rather than "We're working on it."

The Concept Editor, therefore, can format and print the information that is entered into. Once again, these documents can also be used to verify with the expert whether the KE's understanding of a particular aspect of the domain is correct.

The Concept Editor is shown in action in figure 10. From left to right and top to bottom the sub-windows contain: a list of all concepts in the domain, a list of all attributes of a particular concept, a text window and a graphics window known as the desktop. The text window is used to show the concept attributes and other text the knowledge engineer wants to edit or view. The desktop is the main means of interaction. The user interface is a mix between the Smalltalk style (paned windows and popup menus) and the Macintosh style (objects represented by icons, clicking and typing as the basic mechanisms).

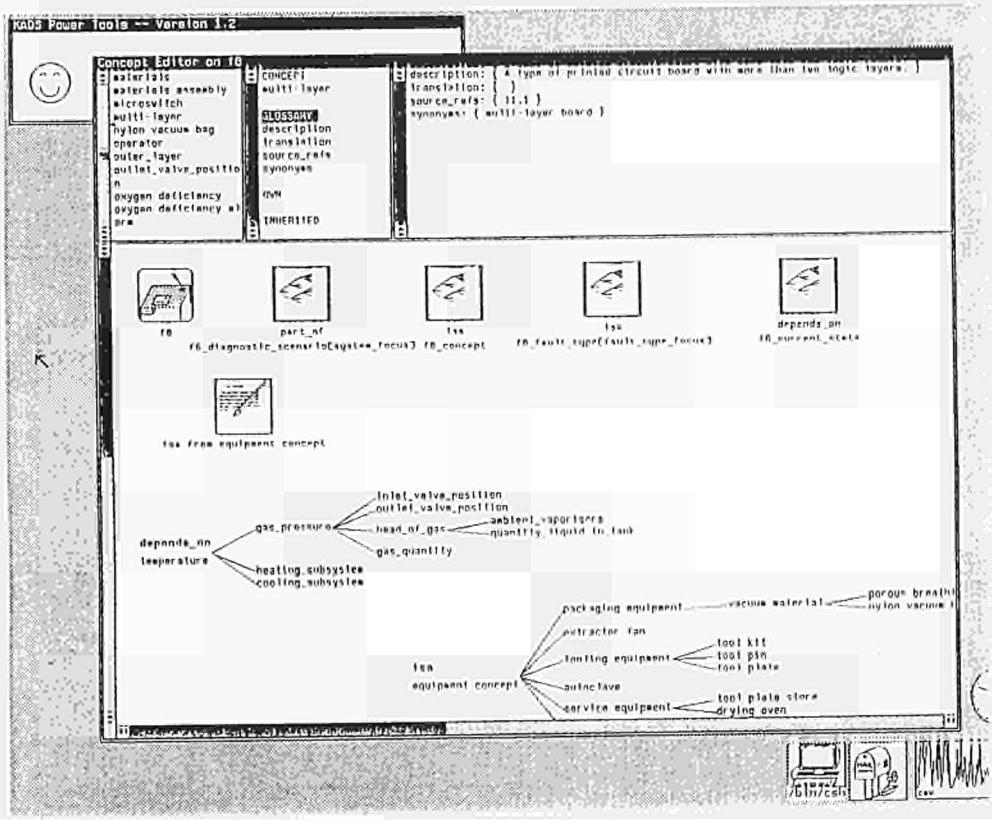


Figure 10 Concept Editor

Conceptual Model Editor

The Conceptual Model Editor and the Interpretation Model Librarian support the epistemological level in knowledge acquisition. The primary function of the Interpretation Model Librarian is to allow the knowledge engineer to choose one or more Interpretation Models that fit the domain. The user may browse through the hierarchy of models and at any point get more detail of the components of a model. The Conceptual Model Editor then allows the construction of a model by combining / modifying / adding to existing models. This includes support for the syntax of the diagrams used.

Environment

The KADS Power Tools are designed using state-of-the-art software engineering and some AI techniques. The system is written in a hybrid programming environment called PCE-Prolog (Anjewierden, 1987), consisting of PCE, an object-oriented programming system and Quintus Prolog running on the Sun Workstation. PCE, written in the C language, was developed in parallel with the original KADS system and features high-level windowing, graphics and text-manipulation facilities. The programmer views PCE-Prolog as a whole, all PCE related facilities are available through four Prolog predicates. The availability of PCE-Prolog contributed substantially to the short development cycle of these tools and is now being developed as a system in its own right. It is interesting to note that PCE, though very much a side issue to the main direction of P1098, has proven so valuable that it is now in use by a significant number of other research projects both within and outside the Esprit Programme. This is an example of cross fertilisation between projects where it is important to see the wider applicability of results even if they are not in the main-stream of a particular project.

6. THE STATUS OF THE METHODOLOGY

In order to provide some reasonably objective assessment of the methodology as it currently stands we return to the list of criteria provided by Wasserman, Freeman and Porcella (1983) for IFIP WG8.1 and originally quoted in the P1098 Technical Annex. They suggest that a methodology for Information Systems development must provide the following:

- coverage of the entire development process
  the KADS methodology as currently defined in Breuker et al (1987) and Barthélemy et al (1987) covers the analysis phase, both with regard to the internal and external views of the system. We can suggest how the models produced by this phase map onto system architectures but the design phase needs much further definition. Nevertheless our experience to date shows that a KADS based analysis aids design even if that is left intuitive (cf Hayball 1986, Rooke and Readdie 1987). The current analysis techniques concentrate on modelling of expertise; the analysis of user-system interaction requirements is ill-defined. This requires tackling from the two perspectives, as for other aspects: internal - user modelling etc, external - human factors. Maintenance issues are not yet addressed, although the separation of knowledge acquisition for analysis and subsequent knowledge base refinement and completion is recognized in the revised Life-Cycle Model. The place and function of prototyping requires further study although some clarification is emerging from the Life-Cycle research and related work in Software Engineering (Budde et al, 1984; NCC,1985).

- enhanced communication among interested parties
  our emphasis, particularly in the modelling work, has been on assisting the specialist software developer. The availability of notations and definitions for intermediate results certainly helps such specialists communicate. Indeed team development of KBS's appears much more viable with such a methodology than with traditional experimental methods. It seems likely however that the intermediate results are not very accessible to non-specialist clients (a common problem with software methods) (cf Rooke and Readdie op cit). Nevertheless the clear definition of the process given by the methodology greatly aids in developer-client communication at a management level, and this is arguably more important than the comprehensibility of the intermediate outputs.

- support for problem analysis and understanding
  we believe the notation of Interpretation Models is a major contribution to this objective, since it enables analysis by refinement rather than in a bottom-up fashion. As far as we are aware this is a unique feature of KADS.

- support for both top-down and bottom-up development
  model-driven analysis is top-down but the analyst is not constrained to this. In particular the KADS modelling techniques can be used in a bottom up fashion and the availability of an appropriate notation supports such an approach. In practice we expect that any analysis will be a combination of top-down and bottom-up approaches.

- support for verification and validation
  no specific attention has yet been given to this point beyond noting that verification and validation requires a visible and defined process. We would therefore claim that the P1098 methodology makes verification and validation possible in a way which is not true of more experimental approaches. The specific "hooks" for verification and validation can only be defined once we have more experience of following the methodology through to implementation. Project tasks to cover these issues are planned in the later stages of the project.

- support for design and performance constraints
  the "external" aspect of Requirements Analysi, makes the *capture* of these constraints explicit. At present the assessment of their impact can only be intuitive, pending the fuller definition of the design process.

- support for software development organisation
  the definition of activities for the Analysis phase allows a partitioning of the development tasks. The lifecycle phases in general allow divisions between activities based on timing and skills required. This has led in at least one case to implementation being carried out in a separate group (physically and organisationally) from analysis and top-level design.

- support for system evolution during its lifetime
  we have concentrated to date on the initial creation of a system. We hope that by the later stages of the project there will be sufficient experience of operational KBS's (both inside and outside the consortium) for relevant data on system evolution to be available.

- automated support tools should be possible
  this is a major feature of the work to date and there has been an important inter-play between the development of tools and the development of the theoretical basis of the methodology. One weakness at present is that it is not clear how the tools should be used to support the top-down or normative aspects of the methodology. We are covering the normative aspects initially in a Handbook and we believe experience in using this will enable us to define the automated support of this aspect more effectively.

- support for software configuration
  we are not directly addressing this point and do not see it as critical for the types of system currently under development.

- teachability and transferability of the methodology
  this has become a major issue within the project, quite apart from the question of transferring results outside. The continual evolution of the methodology has made teaching it difficult. However we believe the analysis component is now sufficiently stable to enable it to be documented in the form of a Handbook. This, together with the support tools, will be the primary means of transferring the methodology to groups within the consortium organisations outside the actual research team. The partners believe that this is a critical factor in the success of the project and within 6-12 months we should have clear indications of success (or otherwise) in transferring the methodology.

- open-endedness of the methodology i.e. open to evolution and development
  it is not clear to us what exactly is meant by this criterion, particularly since one might expect a methodology to have a high degree of stability given the organisational constraints involved in introducing and maintaining a methodology. Nevertheless there can be no doubt that the P1098 methodology is open-ended at present so we must presume to satisfy this criterion.

Acknowledgements

The results of an Esprit project are very much a team effort, so it is invidious to single out particular individuals. A report like this is very much a compilation of project results and includes material presented in greater detail in many other project documents. I thank all my co-workers for their efforts which have made this a fruitful project.

7. References

Allen J., and Anjewierden A. "KADS Power Tools: User Interface Specification" Esprit P1098 Working Paper 1987

Anjewierden A. and Allen J. "KADS Power Tools: User Guide" Esprit P1098 Working Paper 1987

Barthelemy S, Edin G, Toutain T, Becker S. "Requirements Analysis in KBS Development" Esprit P1098 Deliverable D3 1987

Breuker J, Wielinga B, van Someren M, de Hoog R, Schrieber G, de Greef P, Bredeweg B, Wielemaker J, Billeaut J-P, Davoodi M, Hayward S. "Model Driven Knowldge Acquisition: Interpretation Models" Esprit P1098 Deliverable D1 1987

Budde R, Kuhlenkamp K, Mathiassen L, Zuellinghoven H (eds) "Approaches to Prototyping" Proc of Working Conference on Prototyping (Namur 1983), Springer-Verlag 1984

Clancey, W J "The Science and Engineering of Qualitative Models" KSL Working Paper No. 86-27 1986

Dearnley PA and Mayhew PJ "In favour of system prototypes and their integration into the system development cycle" The Computer Journal 26(1) 1983

Freiling M, Alexander J, Messick S, Rehfuss S, Shulman S. "Starting a Knowledge Engineering Project: A Step-by-step Approach" AI Magazine 6(3) 1985

Hayball C.C. "KADS and Object Oriented Design" Esprit P1098 Working Paper 1986

Johnson, P & Gruber, S "Specification of Expertise: Knowledge Acquisition for Expert Systems" AAAI Workshop on Knowledge Acquisition for KBS, Banff Canada, 1986

Littman D. "Modelling Human Expertise in Knowledge Engineering: Some Preliminary Observations" AAAI Workshop on Knowledge Acquisition for KBS, Banff Canada, 1986

NCC "Prototyping" NCC 1985

Nelson T. "Replacing the Printed Word: A Complete Literary System" Information Processing 80, IFIP, North Holland 1980

Rooke P, and Readdie M. "A Study in the Commercial Application of the KADS Methodology" Esprit P1098 Working Paper 1987

Wasserman A.I, Freeman, Porcella "Characteristics of Software Development Methodologies" in (eds) Olle T.W, Sol H.G, Tully C.J. "Information Systems Design Methodologies: A Feature Analysis" North Holland 1983

Wielmaker J. "The Design of KLASS" Esprit P1098 Working Paper 1987

APPENDIX - INTERPRETATION MODELS IN USE

Introduction

The following example shows the use of Interpretation Models in the course of a development project conducted for an STC factory. The system is intended to advise machine operators in one of the process shops on what to do in the event of various alarms being triggered. It is based on the expertise of one of the industrial engineers in the factory who was responsible for the installation of the equipment and currently advises on problems in its use.

This analysis was undertaken by L. Land and T. Mulhall of the KBSC, Polytechnic of the South Bank and I. Wright and C. Hayball of STL.

It will be seen that the development of the model revolves around refinement of the Inference Layer. While this is not necessarily the case it is a common pattern. The final model resulting from this development of course includes a lengthy definition of the domain layer and detailed definitions of all the meta-classes and knowledge sources. These are not included here.

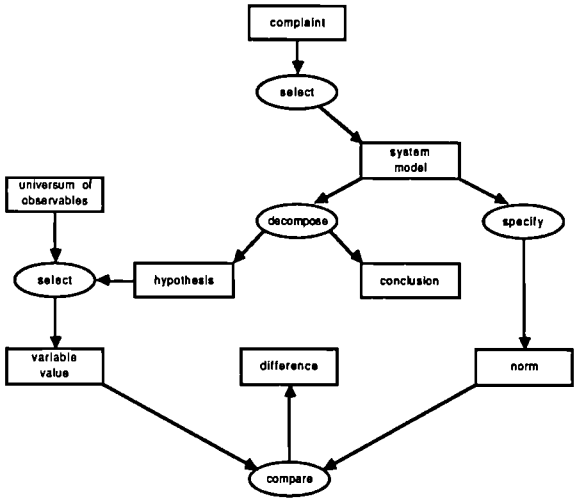
Determining the generic tasks

The current library of Interpretation Models (cf. Fig 7) is divided into two sets of models: analytic and synthetic. It is clear that the task we are dealing with is analytic, i.e. single solutions exist: In several examples of problem solving, the expert was able to give a single solution to the fault reported by the operator. Of the analytic tasks described in the library, a diagnosis one is the most appropriate for our purposes. The expert is required to provide a diagnosis of the fault reported by an operator. This preliminary elimination procedure enables us to reduce the consideration of generic tasks from the library to only three, namely 'localisation diagnosis', 'causal diagnosis' & 'heuristic classification'. Multiple fault diagnosis is also a likely model, but the library does not at present give any details.

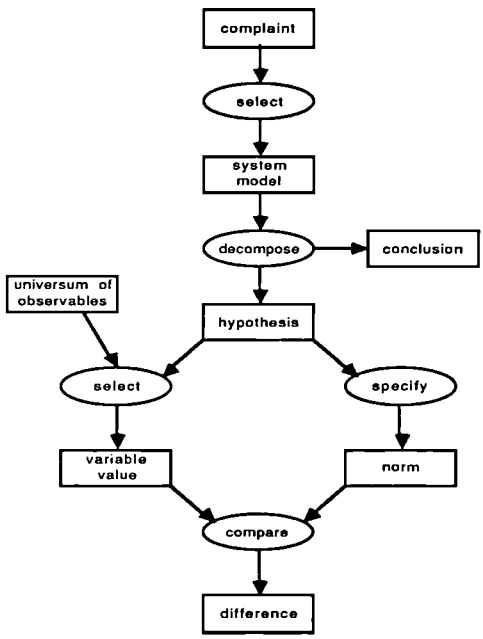
It was at that stage that the transcripts did not seem to fit adequately into any of the structures given by the interpretation models of the above three generic tasks. For example, the expert may sometimes be able to match a complaint directly to a solution (perhaps as a result of having come across the situation before), but it is often the case that he cannot give an answer straightaway and has to go through the set operating procedure with the operator in order to locate the fault. The latter example seems to imply a localisation task but the former is a heuristic classification task. Of particular difficulty are situations when the expert would follow the cause of a fault and then do a localisation task in order to find what when wrong. It is not clear which generic ask follows another, indeed it seems more likely that one is embedded within another. Thus a new interpretation model must be developed in order to guide the analysis in a coherent way. This new model is derived by integrating the above diagnostic models, with the possibility of creating new metaclasses and knowledge sources. We term it 'mixed-mode diagnosis'.

Integration of Localisation, Causal Tracing, Refinement Interpretation Models

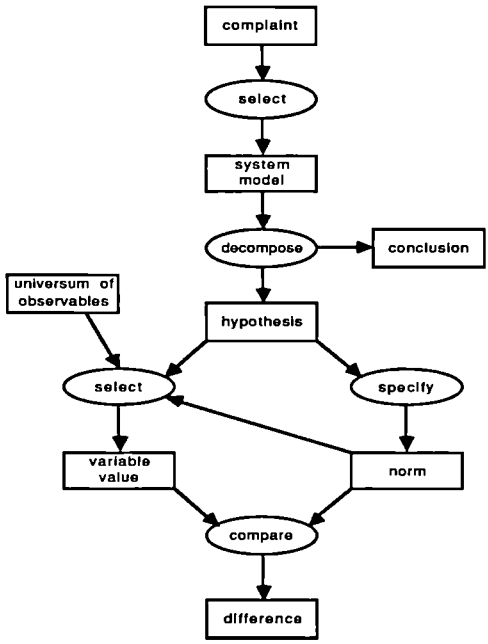
The following diagram represents the inference structure for both localisation and causal tracing. The difference in each case, is the role in the reasoning process of the meta-classes. In Localisation the system model will be a consists_of hierarchy. In Causal chaining the model will be a causal inference net. Implications of the type of system model may be extrapolated throughout the rest of the inference structure. For example, the type of support knowledge required by the decomposition knowledge source will be different in each case, the type of hypothesis that will be generated will be different, etc. The inference structure will also support Refinement if the system model is an is_a hierarchy.



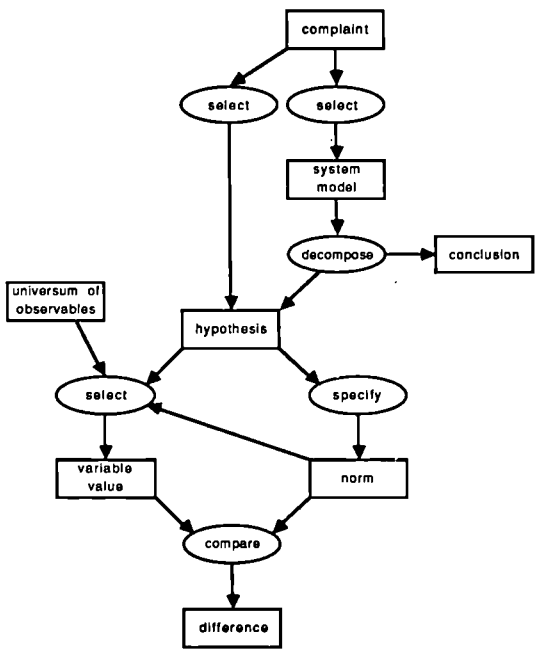
In our considerations of this inference structure we noted some problems. Firstly, we thought that a norm should be selected on the basis of the current hypothesis and not the system model. The norm should be able to tell us something about the value of variable selected on the basis of the current hypothesis. Hence, we made a change to the inference structure giving the following:-



An additional problem was noted here. The variable value to be selected from the universum of observables must be comparable with the norm. Hence, it would make sense if the norm were taken into account in selecting the variable value. This gives us the following:-



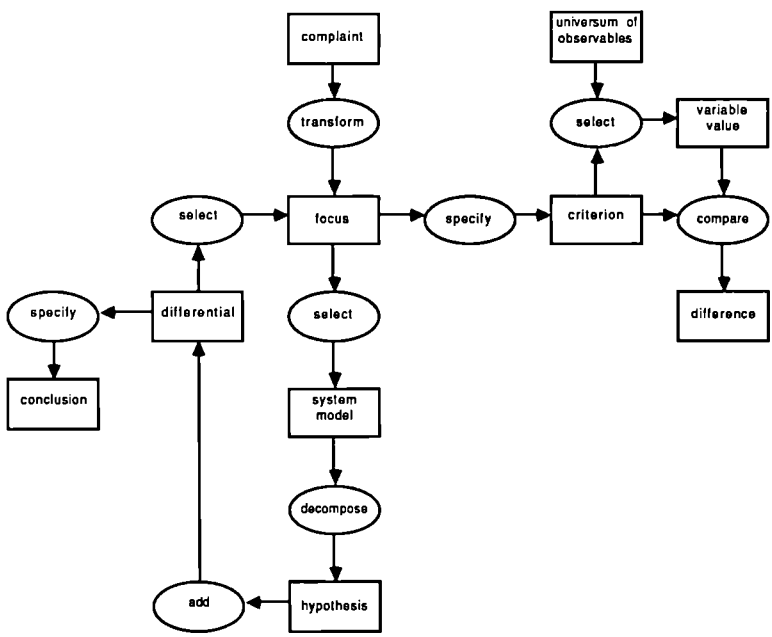
The previous changes that have been made were general observations about the inference structure of a generic task on offer from the library. They do not relate specifically to the current domain of study. A problem which does is the inability of the inference structure to support the prioritising of candidate hypotheses. This is necessary because there is a requirement to give priority to hypotheses relating to potential risk to operator safety. We may facilitate this by selecting, using support knowledge, a hypothesis upon the basis of the complaint, as follows:-



This represents a recursive inference structure in which the hypothesis at one level of invocation may become the complaint at the next. It may represent localisation, causal tracing or classification depending on the nature of the system model. It represents a combination of all three, if we take into account the possibility that each time a system model is selected it may be based on a different organisational principle to the previous selection (e.g. causal rather than consists_of). Although, for the latter to be the case, different system models would have to be available.

Unfortunately this transgresses a "rule" within the methodology that a metaclass may be produced by only one knowledge source. The underlying problem here is that the hypothesis metaclass has at least three roles with respect to the kinds of inference that it is desirable to make. The three roles are as follows; a particular hypothesis, a set of current hypotheses and a particular hypothesis currently receiving consideration. We considered it preferable that the name of a metaclass should relate explicitly to its role in inference. Hence, we have introduced two new metaclasses (differential and focus) relating to the latter two of the roles mentioned above. We can now clearly see the root of the representational problem in the previous version of the inference structure. Whereas we decompose a system model into (a set of) hypotheses, we need to select a particular hypothesis for consideration (the focus), from the set of current hypotheses (the differential).

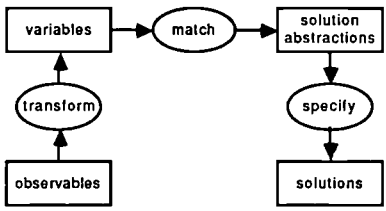
The resultant inference structure, shown below, facilitates the features described above, yet still displays features inherited from its ancestors. Two additional changes have been made. Firstly, we are interested in the potential of diagnosing multiple faults. Hence, a single conclusion may be specified from the differential, still leaving other hypotheses current. Secondly, to provide additional flexibilty we would prefer to be able to both deny a focus or to verify it. We use criterion (instead of norm) as a new metaclass to reflect this.



This still has two arrows going into a metaclass (focus). The difference between this and the previous offering in this respect is that here it represents two alternative routes through the inference structure, whereas previously it represented the ambiguity of the hypothesis metaclass with repect to its role in inference.

Subsumption of Heuristic Classification

So far we have discussed systematic diagnosis by both localisation and causal tracing. We have also discussed the notion of refinement classification as an add-on extra. What of heuristic classification? If there is a requirement, how may this be accommodated within our present inference structure? The inference structure for heuristic classification is as follows:-



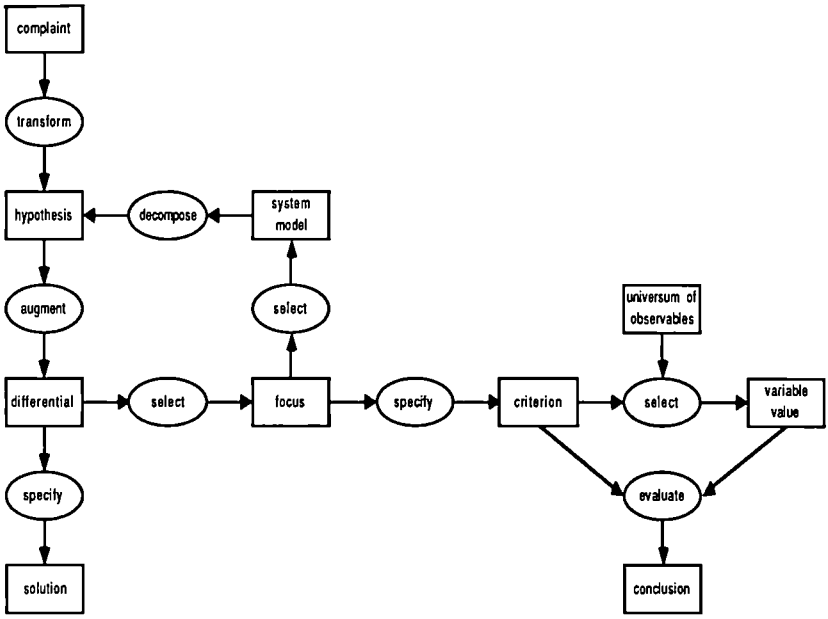
At first glance this seems to be very far removed from our present inference structure. However, on closer inspection we may see that a mapping of some sort may be achieved. For heuristic classification a complaint may take the role of an observable. The focus may take the role of a variable. Everything from system model, following the arrows round, to differential, may be taken together as a composite solution abstraction. The system itself may be viewed as an heuristic model, and decomposition may be viewed as heuristic decomposition. Instead of specifying a conclusion we are matching a solution. In effect all that we are doing is implying that there is a requirement for richer support knowledge for some of the knowledge sources, and through this we can accommodate heuristic classification in addition.

We may use support knowledge to select a focus and to select a system model. Strategy then becomes subsumed within these two (different) knowledge sources. This obviates the need for a process structure at the strategic level.

Further Refinements

Additional observations were made at this stage. It seemed more descriptive to use "augment" as opposed to "add" when putting a new hypothesis together with those already existing in the differential. The criterion may be a complex formula, rather than something as simple as a tolerance. In which case producing a conclusion by the evaluation of a criterion with respect to a variable value is more natural and more flexible. That being the case, conclusion in the previous offering will be renamed solution. In addition, it is difficult to see how the present arrangement could handle situations in which we have more than one complaint. We may transform them into foci, but it is difficult to see how we may prioritise them. Instead we have chosen to transform complaints into hypotheses which are used to augment the differential. These may now be selected as a focus, priority being given by the support knowledge of the select knowledge source.

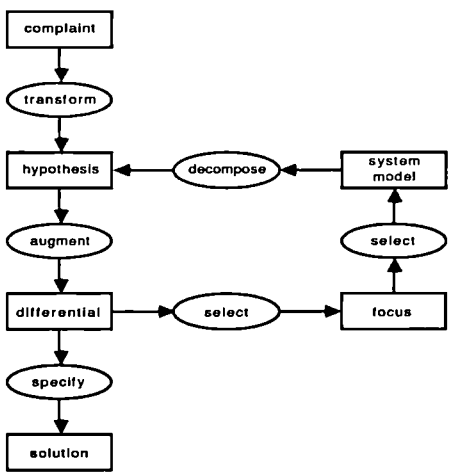
This gives the following:-



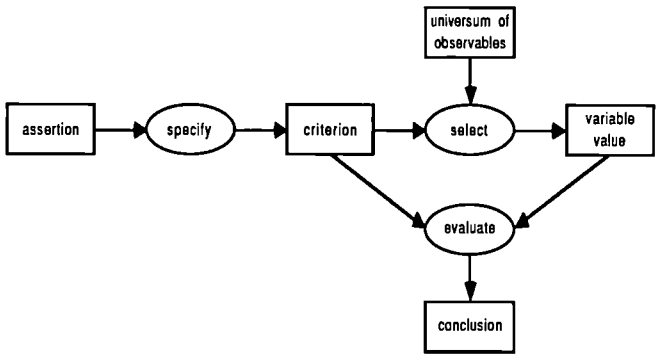
Clearly, what we have now described in the right half of the inference structure is verification of the focus. In addition, we may also wish to verify a symptom before transforming it. This implies having a complete copy of the right hand part of the inference structure stemming from the complaint metaclass. It seems more convenient to view verification as a new generic task. This being the case we are now back in the situation of having two inference structures, one for mixed mode diagnosis and one for verification. We show these structures in the following section. We also offer a task structure. Note that verification is used as a kind of inferential sub-routine by mixed mode diagnosis. Hence, the two tasks are linked at the task level. The requirement for a strategic level is removed.

Final Inference Structure

(1) Mixed Mode Diagnosis



(2) Verification



Task Structure

```
diagnose (complaint) by
    create differential by
        for each symptom do
            verify (symptom),
            transform (symptom),
            augment (differential),
    refine (differential) by
        until refined do
            select (focus),
            classify (focus);
            verify (focus);
            (select (system model), decompose (system model), augment (differential))
    specify (solution)


verify (assertion) by
    specify (criterion)
    until conclusion reached do
        select (variable value),
        obtain (variable value),
        evaluate (conclusion)


classify (hypothesis) by
    for each problem category until classified do
        verify (the hypothesis has this problem category)
```

Project No. 1117

THE DESIGN OF AN INFORMATION RETRIEVAL ASSISTANT SYSTEM

Gert Schmeltz Pedersen and Henrik Legind Larsen

Dansk Datamatik Center
Lundtoftevej 1C
DK-2800 Lyngby, Denmark

Abstract: The KIWI project (ESPRIT 1117) is developing a knowledge-based, user-friendly system for utilization of information bases. To evaluate the KIWI system we are performing a case study by developing a knowledge-based system, which we call KIRA for "KIWI Information Retrieval Assistant".

The aim of KIRA is to assist the information user as a human intermediary would do, hiding to some degree the confusing dissimilarities between different information bases, such as access protocols, query languages, thesauri, and other facilities offered by suppliers of information services.

The paper describes the specification and design of KIRA including the necessary knowledge as acquired from human intermediary experts, as the basis for programming KIRA in the object-oriented knowledge representation language, called OOPS, developed as part of the KIWI project. Much of the work in the project is not in the focus of this paper, and therefore only mentioned briefly.

1. INTRODUCTION

Today's information users are offered a large amount of information in online accessible form. This includes information available from thousands of public information bases and from the bases in the users' own organization. Although an efficient utilization of available information is an important part of good decision making, such utilizations are often not attempted. It is simply too difficult for the user to overview the available information bases, to identify the bases that may contain the information needed, and to learn the particular query language and the specialities of each host supplying retrieval services from the information bases.

Efficient utilization can only be expected, when the information user finds that the resulting improvements in decision making is likely to justify the costs of such a utilization. These costs may include the employment of an information retrieval intermediary, i.e. a person, who has the expertise needed for an efficient utilization of information bases.

The aim of the KIWI Information Retrieval Assistant, KIRA, is to provide the user with a software system that offers some of that expertise. KIRA will support retrieval both from administrative relational databases that are typically connected to KIRA via local area network (LAN), and from bibliographic databases that are typically connected via telephone.

It is characteristic of KIRA that it has knowledge of how to create a representation of user needs, and of how to plan and modify searches in databases, including selecting among several attached databases. Much of this knowledge is represented in rules, as usual in knowledge-based systems. In KIRA the rules are grouped in rulesets, which are hierarchically ordered, such that the root ruleset, the metarules, will determine when to use the other rulesets. This is a convenient and modular way of exploring different interview, search, and feedback strategies.

This paper is organized as follows: Section 2 contains a model of the information retrieval task, discussing its realization in different contexts. In section 3 we take a look at some of the efforts to provide computer-based assistance to the information retrieval task; for this purpose we propose a taxonomy of such systems. Then we are ready to give, in section 4, a description of the KIWI system, which is the basis for the design and implementation of the KIRA system. In section 5 we present the intended functionality of the KIRA system, using the taxonomy to characterize it. Section 6 then contains the design in broad outline. Finally, we provide a conclusion.

2. INFORMATION RETRIEVAL (IR)

In the literature [1] we can find an illustration of the field of information retrieval as given in figure 1.

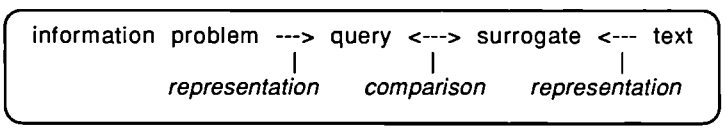


FIGURE 1
The information retrieval task

In principle, an information problem is tackled by comparing a query, which is some representation of the information problem, with surrogates of texts, retrieving the surrogates that match with the query; and then probably retrieving the texts represented by these surrogates.

In the pre-computer world of libraries, the texts are books or other kinds of documents, the surrogates are card indexes, the query is expressed in natural language, and the comparison is carried out by the information users, or by librarians on their behalf, by consulting the indexes. These may exist in several versions, one indexed by author name, one by title, and one by subject.

In the world of computerized, commercial information retrieval systems, texts and text surrogates are stored in databases as text files and formatted records. The databases are operated by host organizations that offer information services, and the most common type is bibliographic databases. A query is expressed in a command language allowing boolean combinations of search conditions on some fields of the records. The comparison is carried out by the host computer. A standard command language is underway from ISO; it is called CCL, Common Command Language [2].

If we turn for a moment to the world of administrative databases, the picture of data retrieval would resemble figure 1. Only the surrogates would normally not stand for texts, but for entities of some universe of discourse. A standard query language for relational databases is underway from ISO; it is called SQL, Structured Query Language [3].

The task of representing texts in surrogates is called indexing. Whether indexing is performed manually by librarians or domain experts, or automatically by computer programs, there is the problem that the surrogate is only a faint reflection of the text, and that it is highly dependent on the indexer. In the administrative database world indexing is often called conceptual modelling.

There seem to be 3 types of information problems or needs [4]:

- "Verificative needs", where the user wants to verify or retrieve information on information items with known characteristics.
- "Conscious topical needs", where the user wants to clarify, review, or pursue aspects of a well-known subject.
- "Muddled topical needs", where the user wants to explore new concepts outside well-known subjects.

The task of representing information problems in queries is not well understood in general. In libraries the librarians, often called intermediaries in this capacity, interview the users trying to understand the information problem and to formulate queries that are compatible with the available indexes. In the case of bibliographic databases the user or his intermediary has to know the relevant databases, their structure and command language and how to express the problem by these commands.

Once the texts are represented and the queries are formulated in a suitable language, the comparison should be straightforward. But according to experience, the results often do not satisfy the user's needs. This calls for a reformulation of the query, whereby the user will employ broader or narrower or related search terms, trying to guess how the indexer might have chosen to characterize the relevant texts. Again, the intermediary may do this by interviewing the user and getting his feedback to the first results. The comparison task may employ a function that maps each surrogate and query into a retrieval status value [5], whereby surrogates can be ranked in order of relevance for the query.

The success of an information retrieval session is traditionally quantified by two measures. The first one is called *precision*, which is the number of retrieved, relevant documents divided by the number of retrieved documents. The second measure is called *recall*, which is defined as the number of retrieved, relevant documents divided by the number of relevant documents. In order to be able to calculate these measures, the user must provide relevance feedback.

3. COMPUTER-ASSISTED INFORMATION RETRIEVAL (CAIR)

Much effort has been, and is being, invested in developing computer programs that can assist in information retrieval. In order to characterize these efforts and our intended KIRA system, we propose the taxonomy for CAIR systems given in figure 2. We do not claim that it considers all aspects and details, only those sufficient for our purpose. We will give comments on a few of the entries now. Other

entries have been discussed or will be discussed with the examples and with the KIWI and KIRA systems.

Among the *searcher groups*, the *elite* user possesses both information retrieval knowledge and highly specialized conceptual knowledge about a particular domain of interest. The *intermediary* has information retrieval knowledge, but limited specialized conceptual knowledge. The *end user* knows about his domain of interest, but has scarce or no information retrieval knowledge. The *layman* has neither specialized domain knowledge nor information retrieval knowledge.

Exact match comparison techniques will retrieve only those surrogates that match exactly with the query. This is typical of SQL. *Partial match* will also retrieve surrogates that have some small semantic distance from the query. The techniques available to the user is dependent on the query language. In CCL only feature based match is available.

We will now use the taxonomy to characterize three CAIR system types.

The simplest of the three types are *gateways* [7], in that they have almost none of the knowledge mentioned in the taxonomy. In the simplest case they only have knowledge of how to access a certain set of databases, enabling them to connect the user to the databases, but from then on the user is essentially on his own, without intermediary help. In more sophisticated cases they may as well know the structure and language of the databases, they may have some subject area knowledge, and they may accept needs specification in a menu-driven fashion.

In the case of *retrieval from an office information system*, such as in the MINSTREL ESPRIT project [8], the CAIR system is connected to the database over a local area network. This implies other time and cost considerations, than when connecting over telephone lines as for gateways. MINSTREL has all the user interface features of the taxonomy except for pseudo natural language, it has most of the kinds of knowledge, and it has comparison techniques of several kinds, allowing ranking of documents and handling of missing or incomplete data. But on the other hand, it does not plan and modify searches as an expert intermediary would do, using user interview and relevance feedback, and it can only retrieve from the local databases that it was designed for.

Now, an *expert intermediary system* such as I^3R [9] would not take the user's own needs specification for granted, but would carry out an interview and use relevance feedback. I^3R can thus take care of all 3 kinds of user needs mentioned in the taxonomy. It has only one database attached to it, but since it is integrated with it, running on the same computer, it can utilize powerful partial match techniques. Other expert intermediary systems could run on the user's personal computer, connecting over telephone lines. This will only allow them the comparison techniques provided by the host system through the query language. The KIRA system is intended to be such a system, except that we will not claim to reach an expert performance level within our scope of resources.

4. THE KIWI SYSTEM

The aim of the KIWI project (ESPRIT 1117) is to design and implement a prototype system (*the KIWI system*) that allows the user to develop knowledge-based applications that make use of data from a number of external databases. The KIRA system is such an application. The applications will be written using a knowledge representation language (called OOPS), that is basically object-oriented, but

• Connection to host database system(s)	• integrated • via LAN • via WAN • via telephone	
• Attached databases	• number	• one • many
	• types	• bibliographic or similar • administrative • statistical, product, ...
	• languages	• CCL or variant • SQL or variant • special
• Searcher groups [4]	• the elite • the intermediary • the end user • the layman	
• User needs [4]	• verificative • conscious topical • muddled topical	
• User interface	• needs specification	• command-driven • menu-driven • term-oriented • browsing the knowl.base • (pseudo) natural language • relevance feedback
	• explanation	
	• result presentation	• textual/tabular • graphical
• Knowledge of [4][6]	• creating representations of documents (indexing) • subject areas (domains) • databases - access, structure, language, costs • users • creating a representation of user needs • planning and modifying searches • improving/learning	
• Comparison techniques [1]	• exact match	
	• partial match	• feature based • structure based • clustering • browsing in surrogates

FIGURE 2
A taxonomy of computer-assisted information retrieval systems

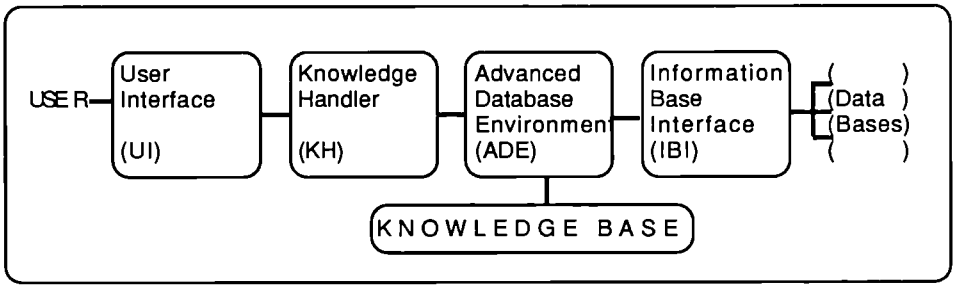


FIGURE 3
The architecture of the KIWI system

enriched with a number of other powerful paradigms such as rule-based, procedural, and logic programming, and monitors.

The KIWI system consists of four software layers and a knowledge base, as illustrated in figure 3. We give a brief explanation here, whereas a longer elaboration is not in the focus of attention of this paper. The architecture was explained in detail in [10]

The *User Interface* (UI) assists users in the interaction with the KIWI system. This software layer is written in OOPS and is devoted to maintain a friendly and intelligent dialogue with users by allowing both their navigation through the KIWI knowledge base and an easy and assisted query facility on the databases attached to KIWI.

The *Knowledge Handler* (KH) implements OOPS and is used to translate user queries into a form acceptable to the Advanced Database Environment, to handle the knowledge on the utilization of databases, and to allow a knowledge engineer to directly encode the knowledge in terms of the problem domain.

The *Advanced Database Environment* (ADE) is responsible for the management of the KIWI knowledge base and for the retrieval of data both directly from the KIWI knowledge base and, indirectly, through the Information Base Interface, from the databases attached to the system. The ADE supports an extension of the relational model and logic programming features to efficiently implement the OOPS concepts.

Finally, the *Information Base Interface* (IBI) is mainly responsible for routing requests from the ADE to the appropriate database and for returning the results in a suitable form to the ADE.

We have used the taxonomy to characterize both the KIWI and the KIRA system, see figure 4. There the KIWI system is characterized by the •-marked terms in plain characters, while the KIRA system has the additional characteristics in bold characters. Characteristics of neither system is °-marked.

The user of the KIWI system could be a knowledge engineer, programming in OOPS, or a searcher with information retrieval knowledge and experience corresponding to an intermediary.

The functionality offered to the searcher is primarily:

- Attachment,
- Refinement,
- Basic search.

Attachment is the process of describing a database to KIWI by giving the schema, the associated names and terminology and other attributes. KIWI accepts attachment of relational databases, typically via local area networks. This provides the knowledge of databases (and their hosts).

Refinement is the process of connecting the user's conceptual model, expressed in terms from his application domain, with the schema terms given during attachment. Refinement results in a network of term relationships, involving user terms, database names, field names, etc. This provides the knowledge of subject areas.

Basic search provides the searcher with the services of the host information system augmented with facilities to log on easily and to utilize the term relationship network in setting up search statements (queries) and in modifying them to be broader or narrower or focused differently. The user needs specification is term-oriented, represented as a subnetwork of the term relationship network, and transformed into SQL by a process similar to one described by Motro [11]. The comparison technique is exact match, and this covers verificative user needs.

5. THE KIRA FUNCTIONALITY

We are not ourselves the specialists in information retrieval. Therefore we have performed a knowledge acquisition process with an expert, Peter Ingwersen from the Royal School of Librarianship in Copenhagen. We have also consulted with Bruce Croft, Amherst. This process involves interviews to elicit the experts' private knowledge and literature studies to acquire the public knowledge. We have not performed verbal protocol analysis and observational studies, as described in [12].

As a result of the knowledge acquisition process we have drawn up the taxonomy, and then used it to define the functionality of the KIRA system as shown in figure 4. The characteristic features are the •-marked terms, including the features inherited from the KIWI system. The additional features are shown in bold characters.

The telephone connection to hosts implies higher communication costs and longer search times, and therefore other search strategies, as compared to more closely connected environments.

The language is to be standard CCL, with an eye open to easy ways of incorporating database specific variants. This implies that more sophisticated comparison techniques than ad hoc feature based are not available from the hosts. One user needs specification could lead to both CCL and SQL queries being generated.

One of the main lessons that we learned during knowledge acquisition was that the users often do not express their information problem properly. Therefore an intermediary, and KIRA, must be able to propose reformulations, in trying to create a representation of the user's needs.

The user of KIRA could be a domain specialist with little information retrieval knowledge. He will be supported in his conscious topical needs by interview and relevance feedback. Muddled topical needs would have to be supported by more general world knowledge, which is out of our scope.

Knowledge of subject areas and databases is input incrementally by the users, as for the KIWI system. But in an organization there may be one more knowledgeable user, who performs this on behalf of his colleagues. Later they may input their own preferred terms to the term relationship network, and thus personalize their own version. Also, the KIRA system may be provided from the implementor with more or less tailored knowledge of subject areas and databases.

6. THE KIRA DESIGN

Design is the choice of architecture, techniques, procedures, and data structures plus resource considerations, in order to realize the functionality. We will give a broad outline of some of the design choices that we have made. A more detailed account of some aspects with a querying scenario is given in [13].

The knowledge representation will make use of all constructs of OOPS:

- **Objects** to model hosts, databases, terms, term relationships, users, sessions, etc. Objects are related by classification, generalization, and aggregation, with inheritance. They have features, which are either references to other objects, or functions (methods) in the form of rules, predicates and procedures.
- **Rules** to model strategies for interview, search, and feedback. Rules are grouped in rulesets and ordered hierarchically. The root ruleset contains the metarules that will determine which ruleset to activate in different situations. Rules are forward chained during evaluation.
- **Predicates** to query the knowledge base, to state inference rules, and to model virtual objects, like views in relational databases. Predicates are backward chained during evaluation.
- **Procedures** to give operations such as performing logon to hosts, transforming answers from the databases to the desired format, or calculating rankings of answers.
- **Monitors** to catch user or session characteristics.

The term relationship network model is based on Motro's idea of a loosely structured database [14]. In this model, term relationships have the triple form (s,r,t), where s is the source term, r the relationship term, and t the target term. For example, (expert system, application of, artificial intelligence) is a term relationship expressing that 'expert system' is an application of 'artificial intelligence'. We consider such triples as term relationship facts. A fact (r,s,t) must be of one of the types: E*R*E and R*R*R, where E is the set of entity terms, i.e. non-relationship terms, and R is the set of relationship terms.

Through term relationships we may represent and interrelate the traditional thesauri relationship types, 'broader', 'narrower', and 'related', the abstraction types of semantic data modelling, 'classification', 'generalization', and 'aggregation', other important types, such as 'equal to', 'greater than', 'inverse of', and 'synonymous to', as well as more domain specific types.

The model further contains inference rules defining the valid inferences on facts. The inferable facts (including the stored facts) comprise the virtual term

• Connection to host database system(s)	° integrated • via LAN ° via WAN • **via telephone**	
• Attached databases	• number	• one • many
	• types	• **bibliographic or similar** • administrative ° statistical, product, ...
	• languages	• **CCL or variant** • SQL or variant ° special
• Searcher groups [4]	• the elite • the intermediary • **the end user** ° the layman	
• User needs [4]	• verificative • **conscious topical** ° muddled topical	
• User interface	• needs specification	• command-driven • menu-driven • term-oriented • browsing the knowl.base ° (pseudo) natural language • **relevance feedback**
	• **explanation**	
	• result presentation	• textual/tabular ° graphical
• Knowledge of [4][6]	° creating representations of documents (indexing) • subject areas (domains) • databases - access, structure, language, costs • **users** • **creating a representation of user needs** • **planning and modifying searches** ° improving/learning	
• Comparison techniques [1]	• exact match • **partial match**	• **feature based** ° structure based ° clustering ° browsing in surrogates

FIGURE 4
The KIWI system is characterized by the •-marked terms in plain characters.
The KIRA system has the additional **bold term** characteristics.
The °-marked terms are characteristics of neither system.

relationship network. As an example, the fact (artificial intelligence, broader than, expert system) can be inferred from the facts:

(expert system,	application of,	artificial intelligence).
(application of,	specialization of,	narrower than).
(narrower than,	inverse of,	broader than).

by use of the inference rules:

(B, Y, A) if (X, inverse of, Y) & (A, X, B).
(A, Y, B) if (X, specialization of, Y) & (A, X, B).

The actual **user interface** will be designed by our partners working on that work package, but conceptually there will be a number of windows, as exemplified in figure 5. The interaction between the user and the system through the windows is partly controlled by the user, partly by the strategies in the rulesets. The user is free to provide more or less of the possible input values. For instance, he may give expected minimum and maximum numbers of answers in the ANSWER FORMAT window, or let the system use defaults.

Also, the user may choose to enter command language sentences directly into the CURRENT QUERY window, or let the system built up the query based on the USER NEEDS REPRESENTATION. This is a subnetwork of the term relationship network connecting the user's selected terms and term relationships. The user can select terms and term relationships by typing into the NEEDS SPECIFICATION window, and by pointing in the TERM RELATIONSHIP NETWORK window and in the RELEVANCE FEEDBACK window. He can also update the term relationship network through the TERM RELATIONSHIP NETWORK window, for instance when he typed in a term that was previously unknown to the system, i.e. it was not in the network.

There are two main **strategies**, one for verificative needs, and one for conscious topical needs. The main difference is that the latter will rank answers based on the importance of query terms, and will propose broader, narrower, or related terms in the process of reformulating queries. The aim is to bring the number of sufficiently relevant answers within the cardinality limits and above the precision and recall thresholds. The system will try to infer the type of needs based on the user profile, the expected cardinality, and the number and exactitude of the given terms and term relationships.

The strategy rulesets will make heavy use of the term relationship network, for instance in **selection of database**. The names of databases and their fields are also in the network, and it is therefore possible to determine, if there are one or more subnetworks with close connection between a database name and the subnetwork representing the user needs. If there are more than one, the user may be asked to make a preference, the system may decide based on heuristics, or all possibilities may be queried in parallel.

There will be variants of the strategies in order to take care of the different time and cost considerations involved for connection to the database via LAN or via telephone, respectively. There will also be variants for different user profiles. For instance, a user wanting low control of the session will not need to see or know about the term relationship network.

We will have to explore alternatives of the strategy variants, before the better choices can be determined. We expect that the ruleset representation will provide a convenient way, because of its declarative and modular form.

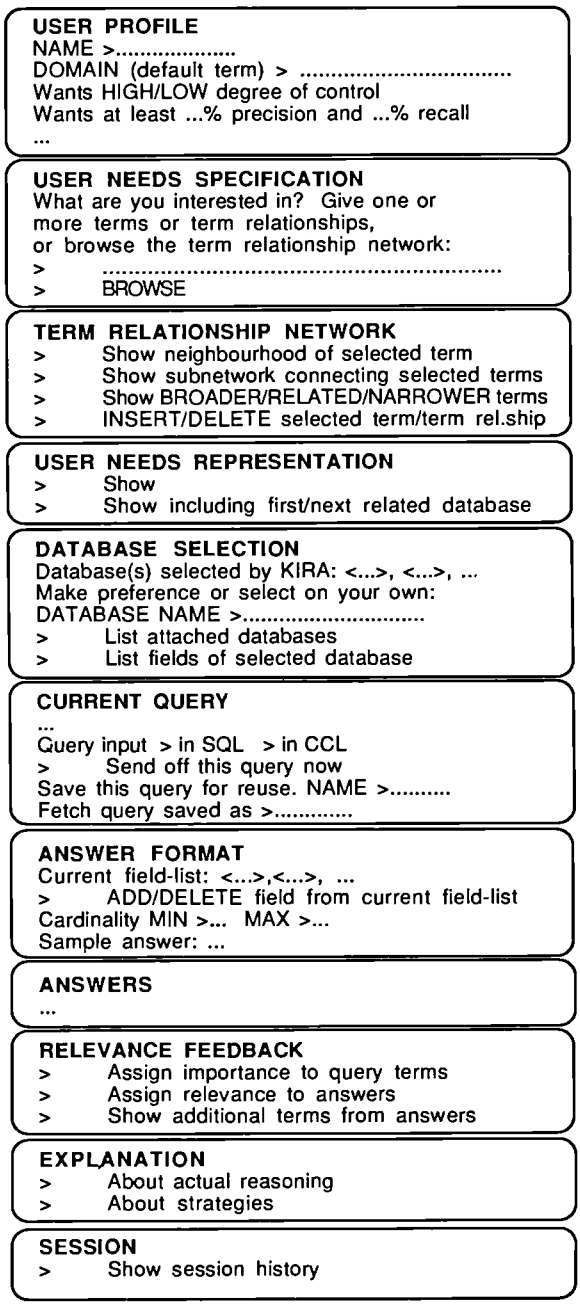


FIGURE 5
User interface, sketched conceptually

7. CONCLUSION

There is a large economic potential in better utilization of the rapidly growing amount of online accessible information. The KIWI project is among the efforts to provide computer-assistance to such a utilization. From many sources it is maintained that employing knowledge-based systems techniques is the way to go. We do that, and we have proposed a taxonomy, that points out how far we go, and that points out in which directions we might extend our scope.

REFERENCES

[1] N.J. Belkin and W.B. Croft, *Retrieval Techniques* (ARIST DRAFT, Rutgers University/University of Massachusetts, 1987).

[2] *Documentation - Commands for Interactive Searching*, ISO/TC46/SC4/N210, 2nd Draft Proposal DP 8777, 1987-03-12.

[3] *Information Processing Systems - Database language SQL*, Draft International Standard ISO/DIS 9075, ISO/TC97, 1986-05-22.

[4] P. Ingwersen, *Cognitive Analysis and the Role of the Intermediary in Information Retrieval*, in: R. Davies (ed.), Intelligent Information Systems: Progress and Prospects (Ellis Horwood, Chichester, 1986), pp. 206-237.

[5] D.H. Kraft, *Advances in Information Retrieval: Where is that /#*&@ Record?*, in M.C. Yovits (ed)., Advances in Computers, vol. 24 (Academic Press, 1985), pp. 277-318.

[6] A. Vickery, H.M. Brooks and B.C. Vickery, *An expert system for referral: the PLEXUS project*, in same as [4], pp. 154-183.

[7] J.J. Hewes, *Gateways to On-Line Services* (PC World, May 1985), pp. 149-156.

[8] G. McAlpine, *Techniques for Filing and Retrieval of Office Information*, submitted to the ESPRIT Conference 1987.

[9] W.B. Croft and R.H. Thompson, *I^3R: A New Approach to the Design of Document Retrieval Systems* (Department of Computer and Information Science, University of Massachusetts, Amherst, 1986).

[10] A. D'Atri, P. Naggar, G.S. Pedersen, D. Sacca, J.J. Snijders, N. Spyratos, and D. Vermeir, *The KIWI System*, presented at the ESPRIT Conference 1986.

[11] A. Motro, *Constructing Queries from Tokens*, Proceedings of SIGMOD'86, May 1986, pp. 120-131.

[12] N.J. Belkin, H.M. Brooks, and P.J. Daniels, *Knowledge Elicitation Using Discourse Analysis*, to appear in International Journal of Man-Machine Studies, 1987.

[13] H.L. Larsen, *Knowledge Representation in IRIS, an Information Retrieval Intermediary System*, Proceedings of the 7'th International Workshop on Expert Systems and their Applications, May 1987 (IRIS was a former name of KIRA).

[14] A. Motro, *Browsing in a Loosely Structured Database*, Proceedings of SIGMOD'84, June 1984, pp. 197-207.

Project No. 415

Overview of a Parallel Reduction Machine Project *

D J Bevan, G L Burn and R J Karia **

GEC Research Ltd
Hirst Research Centre
East Lane
Wembley
Middx. HA9 7PP
United Kingdom.

ABSTRACT

ESPRIT Project 415 has taken what are considered to be good programming language styles and is developing parallel architectures to support them. Here we describe the part of the project which is developing a distributed memory architecture for functional languages.

Designing parallel architectures for evaluating functional languages presents many challenging problems. Firstly a model for the parallel reduction of such languages must be found. An abstract interpretation has been developed which leads to a parallel reduction model. It can be implemented in a compiler so that programs can automatically be annotated with parallelism information.

The original COBWEB, a novel distributed memory architecture, is described, along with the conclusions we have drawn from our simulation work. We also briefly describe some of the architectural features of the architecture we are designing to support the parallel reduction model.

Many programming languages including functional ones require automatic storage allocation which has to be garbage collected. We present another piece of work from our project which has resulted in the discovery of a distributed reference counting garbage collection algorithm which has very low overheads.

* Research partially funded by ESPRIT Project 415 : Parallel Architectures and Languages for AIP - A VLSI-Directed Approach.

** Electronic mail addresses of the latter two authors are geoff@gec-rl-hrc.co.uk and karia@gec-rl-hrc.co.uk respectively.

1. Introduction.

ESPRIT project 415 is investigating several styles of programming languages and architectures to support these languages. The project is taking a "languages first approach" to architectural design. Rather than designing a machine and then trying to implement a language on top of it, the project has chosen some styles of programming languages which are thought to be easier to program in and which are amenable to formal analysis, and is then investigating how to design parallel architectures to support them.

This paper deals with the parallel implementation of functional languages, work which is being completed by GEC Research Ltd at the Hirst Research Centre. We have found it especially interesting in this part of the project to see how much the language style influences architectural design.

The natural reduction model for the λ-calculus, upon which most functional languages are based is a sequential one. Therefore one of the first things we had to do was to see where we could obtain parallelism in the evaluation of functional languages. Our work on the abstract interpretation of functional languages has led us to a parallel reduction model which has the same feeling of naturalness as lazy evaluation does for sequential machines. This work is described in the second section of the paper.

While the work on the parallel reduction model was being completed, we investigated, using simulation, a novel distributed memory architecture for combinator reduction called COBWEB [Hankin, Osmon and Shute 1985]. Changes were made in the abstract machine to incorporate some of the early work on the analysis of functional languages for parallelism information. After a brief description of the architecture, a summary of our conclusions from this work is presented in the third section. We also give some indication of the constraints that a distributed memory architecture place on parallel reduction.

Finally, many languages, including functional ones, require automatic storage allocation and collection. This is a problem which is hard enough for sequential machines, but is even worse for distributed memory architectures. In section four a distributed reference counting garbage collection algorithm is outlined. It is more fully described in [Bevan 1987], a paper which also appears in this volume.

2. Parallelism in the Evaluation of Functional Programs.

There are two broad classes of ways we may choose to obtain parallelism in the evaluation of functional languages which have no explicit parallel constructs. A machine may employ *speculative parallel evaluation*, where all possible redexes in a graph are

reduced in parallel, or it may use *conservative parallel evaluation*, where it only evaluates an expression if it knows it will need its value.

Speculative parallelism wastes machine resources by evaluating expressions which may eventually be discarded. For example, in the expression

if condition then e_1 *else* e_2

the value of only one of e_1 and e_2 will be needed, depending on the truth of the *condition*. The problem is compounded in languages which allow the writing of expressions denoting infinite computations, for such computations may try and consume infinite amounts of resources.(*)

Because of the wastage of resources and the difficulties we foresaw in trying to garbage collect infinite processes, we decided to see if we could find out at compile-time when functions would definitely eventually need to reduce any of their arguments.

2.1. Determining Parallelism Information from Functional Programs.

By only ever evaluating the left-most outer-most redex and evaluating expressions only as far as head normal form, lazy evaluation ensures that no expression is evaluated more than is needed to produce the result of a calculation. While this is perfectly satisfactory for a sequential machine, it is hardly useful for a parallel machine for it only ever allows one expression to be evaluated at a time. The problem is that lazy evaluation is overly pessimistic about which expressions are going to be needed - it only knows that the left-most outer most redex is needed.

Another way of looking at lazy evaluation is to notice that it never initiates a non-terminating computation unless the semantics of the original expression to be evaluated was undefined, that is, bottom. Our problem then reduces to ensuring that we do not initiate a non-terminating computation in evaluating a subexpression unless the semantics of the original expression is undefined. We will call this our *semantic criterion*.

By giving a different interpretation, an *abstract interpretation*, to the symbols in a programming language, we are sometimes able to find out information about a program without running it. An abstract interpretation for determining the definedness of functions in terms of the definedness of their arguments has been developed in a series of papers. Mycroft [Mycroft 1981] developed a strictness analysis for first-order functions

(*) An infinite computation does not necessarily mean no result is produced. When one has structured data types, a computation may produce a finite or unbounded amount of output as well as proceeding forever.

over atomic data types(*). This was extended in [Burn, Hankin and Abramsky 1986] to a strictness analysis for higher-order functions. Wadler [Wadler 1987] introduced an abstract domain for structured data types such as lists. All of these various strands were drawn together in [Burn 1987a] where a framework for the abstract interpretation of functional languages was developed and applied to this problem.

Traditionally this abstract interpretation has been used to give a *strictness analysis* of functional programs, that is, finding if a function application is undefined when one of its arguments is undefined. However, this loses information, for it only tests the semantic criterion locally. By looking at the abstract interpretation in another way, we are able to determine more parallelism information, which leads to a natural model for the parallel evaluation of functional languages. We call the results of this analysis *evaluation transformers*, for they tell how much evaluation can be done to an argument of a function given that we can do a certain amount of evaluation of the function application.

The analysis can be completed by a compiler.

2.2. Evaluators.

We know that in some function applications, the argument will need more reduction than just to *head normal form (HNF)*. For example, an application of the function

$$\begin{aligned} &\mathit{length}\ [\,] = 0 \\ &\mathit{length}\ x{:}xs = 1 + \mathit{length}\ xs \end{aligned}$$

will eventually need to traverse the whole of the argument list, but will not need any of the values of elements of the list. The function

$$\begin{aligned} &\mathit{sumlist}\ [\,] = 0 \\ &\mathit{sumlist}\ x{:}xs = x + \mathit{sumlist}\ xs \end{aligned}$$

needs to traverse the whole of its argument list and also obtain the values of the elements of the list. We will call the process of recursively evaluating the second argument of *cons* until we reach *nil* (if we do, which will only happen if the list is finite), creating the *structure* of the list. There is a similar idea for all recursively defined types, such as integer binary trees which have type equation :

$$\mathit{tree} \cong 1 + \mathit{num} \times \mathit{tree} \times \mathit{tree}$$

or in a Miranda [Turner 1985](**) definition :

(*) An *atomic data type* is one which has a flat domain as its usual interpretation. Integers and booleans are two examples.

tree ::= *NIL_TREE* | *NODE num tree tree*

where the second and third arguments to the *NODE* constructor are recursively evaluated. Basically, evaluating the structure of an expression is unfolding the recursive part of the data type definition.

We will say that we can evaluate an expression using a particular *evaluator*, and call an evaluator which evaluates expressions to HNF ξ_1, an evaluator which evaluates the structure of a list ξ_2, and an evaluator which evaluates the structure of a list and every element of the list to HNF ξ_3. For completeness, the evaluator ξ_0 does no evaluation. The relationship between the evaluators is

$$\xi_3 > \xi_2 > \xi_1 > \xi_0$$

where the relationship > is read as *stronger than*, because the first evaluator does more evaluation than the second.

The abstract interpretation of [Wadler 1987], [Burn 1987a] is able to detect situations when functions need to do more evaluation of their arguments than just to HNF. It can be used to determine *evaluation transformers* [Burn 1987a], [Burn 1987b] which will tell us which evaluator we may use for the argument in an application when given the evaluator we can use for the application.

2.3. A Model for the Parallel Evaluation of Functional Languages.

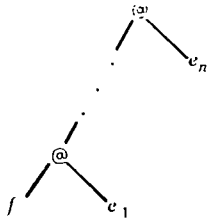
Just as lazy evaluation is the natural model for the sequential evaluation of functional languages, programs annotated with evaluation transformers lead to a natural model for their parallel evaluation.

One way of representing a functional program is as a graph [Wadsworth 1971] of binary apply nodes. Thus the application

$$f\, e_1 \cdots e_n$$

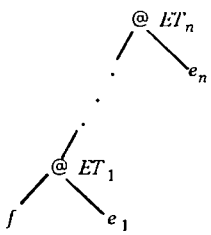
would be represented as :

(**) Miranda is a trademark of Research Software Ltd.



Left-most outer-most reduction is obtained by traversing the spine of the application, until the function f is reached. A copy of the body of f is made, substituting pointers to the arguments e_1 to e_m if f needs m arguments, and the root of this graph overwrites the the application node which points to e_m. Traversal of the spine begins again at this node. An excellent coverage of graph reduction is given in [Peyton Jones 1987].

Suppose we label the graph with evaluation transformers ET_1 to ET_n as in the diagram :



and that we are evaluating the expression with the evaluator ξ. Then the only thing that changes with the evaluation mechanism is that when traversing the spine of the graph, a task is initiated to evaluate each expression e_i with the evaluator $ET_i(\xi)$.

There are some important things to note about this evaluation mechanism. Any particular expression is being evaluated using left-most outer-most reduction. The evaluation mechanism is not some sort of parallel-innermost reduction strategy (i.e. a parallel version of call-by-value), for in the case that the f in the above example is a user-defined function, the evaluation of the expression e_i proceeds in parallel with the evaluation of the expression $f\ e_1 \cdots e_n$.

The abstract machine of [Clack and Peyton Jones 1986] solves the problems of synchronisation of processes evaluating pieces of the graph on a shared memory architecture when only the evaluator ξ_1 is being used. In [Karia 1987] an abstract distributed memory architecture is defined which fully supports the evaluation transformer model of parallel reduction. This is in the process of being further refined [Bevan et al 1987].

3. Parallel Graph Reduction and Distributed Memory Architectures.

For our purposes, we can divide parallel architectures into two broad classes. Shared memory architectures have many processors sharing a common memory. The problem with such architectures is that the memory becomes a bottle-neck in the system.

A distributed memory architecture consists of a series of *processing elements (PEs)*, each containing, in its simplest form, a processor, some memory and communications capacity, connected together by a network. Given a suitable network, a distributed memory architecture in theory has no bounds to its extensibility. Because of this, we have chosen to develop a distributed memory architecture to support our parallel graph reduction model.

Concurrently with the work on the parallel reduction model, we have been investigating a particular computer architecture, the COBWEB, and the next section describes the results of that investigation.

Since finishing the simulation work, we have stepped back in order to determine more abstractly the essential features for the support of parallel graph reduction on a distributed memory architecture [Bevan et al 1987]. Two main findings of this investigation are summarised in section 3.2.

3.1. The COBWEB

One of the architectures that has been investigated in detail in our subproject is the COBWEB, a proposed architecture that exploits the potential of Wafer Scale Integration to support an SKI-combinator reduction model. A description of the architecture is given in [Shute and Osmon 1985] and an abstract machine for it is described in [Hankin, Osmon and Shute 1985].

In this section, a brief description of the architecture along with some discussion of the simulation results is given.

3.1.1. Overall Architecture

The COBWEB consists of a large matrix of identical processing elements on a wafer, each of which is capable of communicating and receiving tokens of machine code to and from its neighbours. The machine owes its name to the way in which the processing elements are interconnected. A bidirectional communications line originating from a central port in the two dimensional matrix of processors traverses through the processors in a spiral pattern and ends at the outermost processor, establishing a circumferential line for communication between processors. Bidirectional lines also

traverse radially from the centre outwards through processors so that tokens can be communicated to an outer level in the web. The configuration is established dynamically by the processing elements on start up, avoiding any faulty processing elements and creating a spiral of all the good ones. This scheme was originally proposed in [Aubusson and Catt 1978], and ensures both fault tolerance and graceful degradation in the event of failing elements on the wafer. An example spiral configuration of processors is shown in Figure 3.1.1-1.

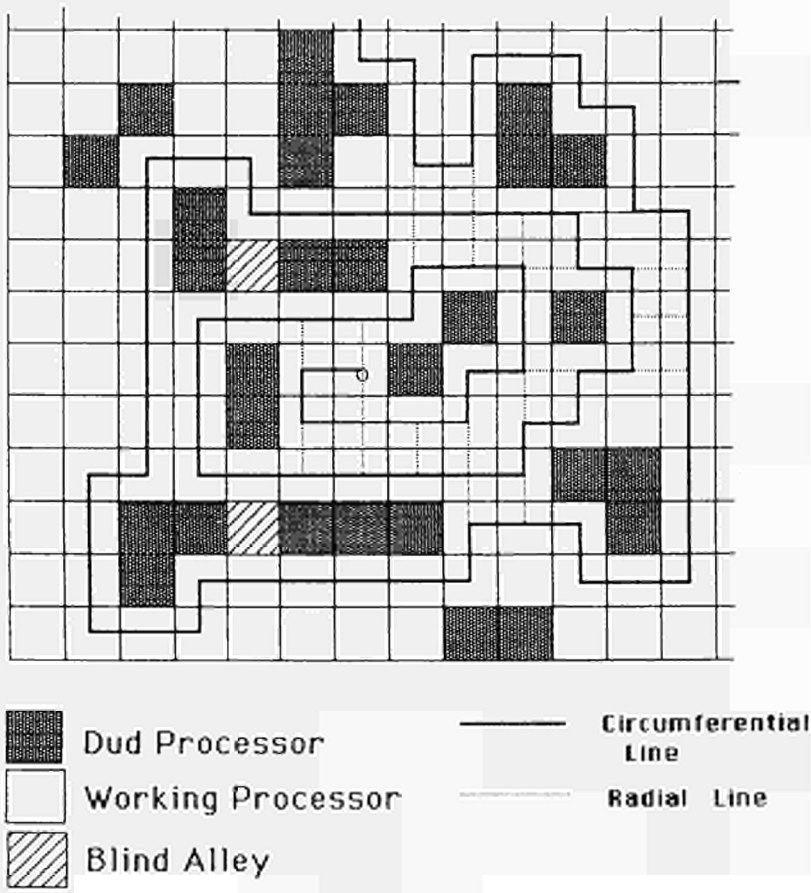


Figure 3.1.1-1

A functional program is translated into combinators which can be represented as a graph. Tokens constitute the nodes of the combinator graph. The combinators used

include the ones used in Turner's machine [Turner 1979] along with the P and P' combinators described in [Hankin, Burn and Peyton Jones 1986] for strict functions. Each token has an identifying tag by which it is addressed by other tokens. A program consists of two types of tokens - those that represent nodes in the function definitions and those that represent the function applications being reduced. Initially, the defining tokens are fed into the machine via the input port. By virtue of the communications logic in each processing element, they arrange themselves in ascending order of tag value along the circumferential line of the spiral, each token occupying one processor. The function applications are then released into the machine to search for their matching definitions. When an application token meets an appropriate definition token, a match occurs, and the latter's body is instantiated and made to replace the tag in the former. If the head of the token is a combinator then the token can be reduced, which may cause further tokens and applications to be produced. These in turn are routed to their respective positions in the spiral. This treatment of tokens continues until results are generated, which are routed to the output port on the outer most circumference. Tokens not referenced by others are automatically destroyed by the garbage collection strategy.

3.1.2. Simulation and Performance Results

The simulation of the COBWEB has been implemented in Interlisp on a Xerox 1108 workstation. It includes an event scheduling simulation model that supports the scheduling of processes representing transactions of both the reduction of tokens and their communication between neighbouring processing elements.

Two major changes were made to the abstract machine given in the above papers [Karia 1987]. Firstly, preliminary results of the work of detecting parallelism information and encoding it in SKI-style combinators [Hankin, Burn and Peyton Jones 1986] was included in the simulation. Early work on abstract interpretation concentrated on strictness analysis, which was interpreted to say when the argument in a function application could be evaluated in parallel with the application. By regarding combinators as directors on application nodes [Dijkstra 1980], [Kennaway and Sleep 1986], this information could be encoded as another director, P, which is semantically like I, but which initiated a parallel process to evaluate the argument.

Secondly garbage collection was inhibited because at the time of the simulation we had no suitable garbage collection algorithm for a distributed memory architecture. This situation has been solved in [Bevan 1987]. As well, deleting tokens in the middle of the contiguous chain of tokens merely increases the number of token hops, an overhead,

whose price obscures the simulated time for the execution of a program.

In the realisation of the abstract machine in the simulation a third change was made in that square rather than hexagonal processors were used.

The simulation experiments were carried out on an "ideal" wafer, i.e. one without dud processing elements in order to determine the maximum possible performance.

We will not go into the simulation results in detail, for they are covered adequately elsewhere, for example [Karia 1987], but briefly discuss some of the lessons which we have learnt from the simulation.

One of the principal results of the simulation of a multiprocessor is its *speedup* factor, i.e. the ratio between the time taken to execute a program on a single processor to that taken on the whole system. This is the scheme adopted by a number of researchers investigating dataflow and reduction machines using simulation techniques, e.g. [Hudak 1985]. The move from a uniprocessor to a multiprocessor allows for parallelism at the cost of overheads in communication, which degrade the ideal performance, viz an n fold improvement (where n is the number of processors). For example a program that takes x time units to execute on a uniprocessor, would ideally take $\frac{x}{2}$ time units on two processors but in practice would take $\frac{x}{2}+y$, where y represents communication overheads. There are, of course, other factors that increase y, such as how much parallelism the algorithm has, memory management within each node, etc. but these are not as readily measurable as communication time. For a distributed memory reduction machine, communication time is dependent on the *access latency*, i.e. the time taken by an expression to access a subexpression whose value it needs. Consequently, simulation experiments are done to investigate schemes on the architecture to minimize access latency. As an example, one factor that influences access latency is the locality of reference in the program.

On COBWEB, the number of reductions done during program execution is merely a fraction of the number of hops. This implies that a fairly coarse grain of computation is required to balance communication overheads before benefiting from migrating work over the spiral. For example, our experiment with *pfac* 1 8(*), where *pfac* is the parallel factorial function defined by

$$\begin{aligned} pfac\ x\ y &= x \text{ if } x = y \\ &= x \times y \text{ if } y - x = 1 \\ &= (pfac\ x\ z) \times (pfac\ (z+1)\ y) \textbf{ where } z = (x+y)/2 \end{aligned}$$

showed that the grain for that program had to be large enough to be at least 6 times greater than the time for one hop. Hence, the need for a larger grain than SKI-combinators

is one of the conclusions drawn from our simulation.

Since communication overheads dominate execution time, the machine's performance largely depends on how efficiently tokens are managed. The experiments that have been done with the simulator have been to identify what values of the variable parameters of COBWEB ensure efficient token management and minimal communication overheads. Unfortunately, the experiments show that suitable values for these parameters depend on the size of the program being executed. For example, the required ratio between ALU time and hop time varies with program size and so does the number of free tags required per cell to avoid tag requests. These, in addition to several other variable factors render the machine's performance is quite unpredictable and hence, no consistent measure of the speedup factor is obtainable. For this reason, the idea to run a suite of programs on the simulation was abandoned. Given the time taken to run a single experiment on the Xerox 1108 (typically 11 and 30 hours for *pfac* 1 8 and *pfac* 1 16 respectively), it was decided that the amount of useful information gained from trying a suite of programs would not be sufficient to justify carrying out such experiments. The results anticipated from such runs is the same as those seen with *pfac*.

One of COBWEB's features that distinguishes it from other distributed memory architectures is its unique addressing scheme. We have attempted to minimize access latency by defining a suitable routing algorithm for a spiral configuration and by providing an element of locality in the distribution of free tags. At present, the routing algorithm relies on the tags of tokens residing in each cell. The latter are constantly being shuffled, and hence, tokens are being sent through a maze of varying addresses to a moving destination. Also, the experiments have shown that radial lines are very poorly utilised. Perhaps allocating absolute addresses to the cells on a wafer would overcome a lot of the problems associated with routing tokens. Having absolute addresses would allow for a packet switching scheme for the routing of tokens as is done in some recently proposed point to point network multiprocessors, e.g. Cosmic cube [Seitz 1985]. It also overcomes the need for free tags, since the address space would be directly mapped onto memory locations or registers in cells. A third advantage of absolute addressing would be that resident tokens would not need to be maintained in a contiguous chain, i.e. gaps in the program are allowable. This does away with the need for pull tokens which are created to close the gaps in the spiral caused by the deletion of a token. A scheme for fault tolerant reconfiguration and token routing on a wafer with absolute address cells has been proposed [Anderson et al 1987].

The use of a spiral configuration makes COBWEB, in the best case (i.e. when there are no faulty cells) a two dimensional grid of processing elements. What is required of a

network in a reduction multiprocessor is to diffuse work rapidly to all processors as it is generated, whilst maintaining locality of reference. The problem of locality of reference in functional programs has so far remained unanswered. However, topologies for multiprocessor systems have been proposed that have the diffusion property, e.g. doubly twisted torus, hypertree etc. These have been experimented with by other researchers in architectures for functional programming. For example, [Hudak 1984] shows how a doubly twisted torus diffuses work better than a complete interconnection of processors when experimenting with a simple diffusion heuristic.

In spite of the defects identified in COBWEB, it has several good features that can be well exploited in reduction architectures. For example, the use of variable sized tokens is a unique feature of COBWEB's abstract machine. The use of variable sized tokens eliminates the need to traverse graphs composed of binary apply nodes as in other proposed machines [Turner 1979], [Johnsson 1987], [Peyton-Jones, Clack and Salkild 1985]. Another favourable feature of COBWEB is the manner in which function definitions are maintained. Since tokens constituting a definition are spread over a number of processors, several tokens that are applications of a definition can concurrently traverse it in a pipelined fashion. This overcomes the need to hold several copies of the definition over the network of processors. Such a scheme is also employed in dataflow machines. Finally, COBWEB has been a very useful vehicle for research, since it incorporates numerous innovative approaches to reduction. Among the ideas generated from working with COBWEB are the refinement of the abstract machine to perform graph reduction and to evaluate programs in parallel (as opposed to normal order), the definition of a distributed reference count garbage collection scheme [Bevan 1987], and some ideas for the definition of an alternative reduction architecture which is described in [Bevan et al 1987].

At present, a collaborative project funded by the Alvey Committee in the UK is under way to investigate modifications to COBWEB's architecture to overcome the aforementioned problems in its original design [Anderson et al 1987].

3.2. Principles for the Design of a Distributed Memory Architecture.

COBWEB embodies many unique architectural features, and is very different to shared memory architectures for graph reduction. With this in mind, we decided to try and find out what features were essential features which had to be included in a distributed memory architecture for graph reduction [Bevan et al 1987].

There were two main conclusions. The first is that at the abstract machine level, programs must be represented as variable-sized tokens. Each token must be a *maximally*

unsharable piece of the spine of the graph. This means that one does not have to traverse the spine over the network and also that sharing is not compromised. Interestingly these tokens correspond exactly to the tokens of COBWEB.

Secondly, the unit of work is to reduce an expression to head normal form. Because expressions are not allowed to be copied until they are in head normal form, this is a natural unit of computation; the root token of the result does not need to overwrite the root token of the original expression until it is in head normal form.

These ideas are incorporated into an abstract distributed memory architecture which supports our parallel reduction model in [Bevan et al 1987].

4. Distributed Reference Counting Garbage Collection.

One of the major problems with implementing languages which require some sort of automatic storage allocation is the recovery of storage when it is no longer needed. For an architecture which has several processors and memories, the problem is compounded because data structures may become spread over several memory modules. Reference counting garbage collection is attractive because memory is freed as soon as an object is no longer referenced, rather than having to stop the machine to do some sort of mark scan algorithm.

A major problem with reference counting garbage collection is that the natural way to think of it working is that whenever a reference to an object is duplicated, then the reference count of the object is incremented. Similarly, when a reference to an object is deleted, the reference count is decremented. This is not easily lifted to a distributed architecture because there is the problem that a decrement message may reach an object before an increment message, and so the object is deleted before it should be. A solution to this problem is to introduce a complex protocol such as in [Lerman and Maurer 1986].

The key point of our algorithm is that it only requires decrement messages to be sent, and so alleviates the need for complex protocols. Below we briefly outline the algorithm which is discussed in detail in [Bevan 1987].

4.1. The Basic Algorithm.

We assume that our machine consists of a number of PEs and that on each PE there are a number of objects. These objects may contain references to other objects (or indeed themselves), possibly on other nodes. At any time a reference may be deleted or a new object may be created and a reference to it created from an object already in existence. Similarly, at any time, a reference may be copied from one object to another. We require

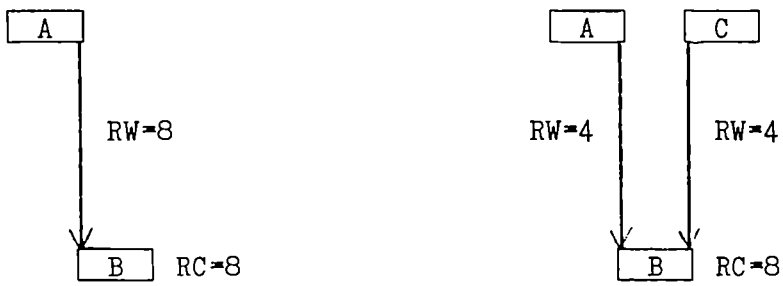
that no object be deleted if it is referenced and that objects be deleted when they are no longer referenced.

In order to achieve this, we associate with each reference a positive weight and with each object a reference count. The algorithm attempts to maintain the following invariant.

> The reference count of an object is equal to the sum of the weights of the references to it.

This ensures that the reference count of an object is zero if and only if it is not referenced by another object. An object can thus be deleted if its reference count is zero. We consider the different possible operations on references separately.

When a new object is created with a reference from some object already in existence, the new object may be given an arbitrary reference count and the reference to it a weight equal to that reference count (see Figure 4.1-1(i)). This maintains the invariant.



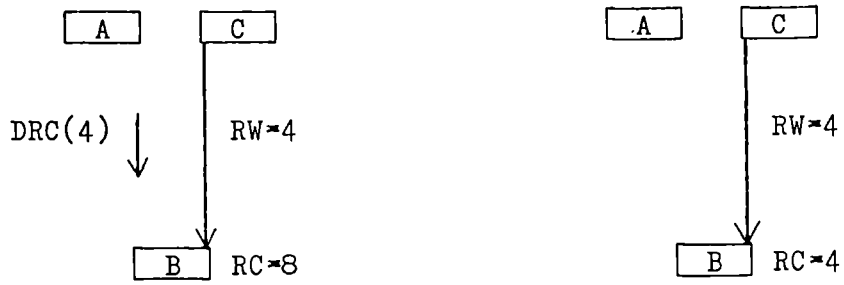
(i) A new object B is created with a reference from A.

(ii) The reference from A to B is duplicated.

Figure 4.1-1

When a reference is duplicated, its weight is split between the two resulting references in such a way that the sum of the weights of the resulting references is equal to the original weight of the initial reference (see Figure 4.1-1(ii)). This maintains the invariant without needing to communicate with the object referenced or to change its reference count.

When a reference to an object is deleted, the reference count of the object needs to be reduced by the weight of the reference. To achieve this, a message, known as a *decrement reference count (DRC)* message, is sent to the object. This message contains the weight of



The reference from A to B is deleted.

Figure 4.1-2

the deleted reference. When an object receives a DRC message, it decrements its reference count by the weight contained in the message (see Figure 4.1-2). Thus the invariant is maintained.

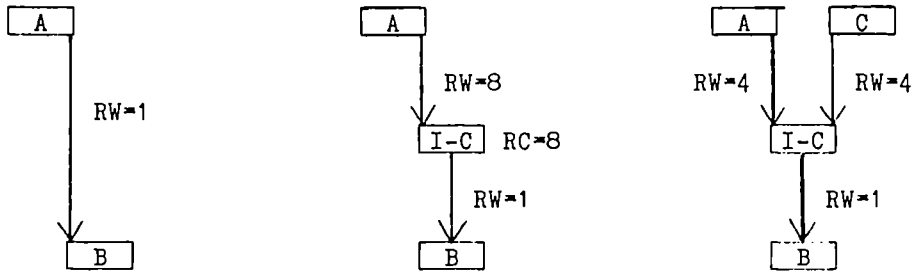
When the reference count of an object reaches zero, the object may be deleted. In order to do this all its references to other objects must first be deleted by sending DRC messages to each object referenced.

The invariant given above now holds if there are no DRC messages in transit. The following invariant holds all the time.

> The reference count of an object is equal to the sum of the weights of the references to it added to the sum of the weights contained in DRC messages in transit to it.

4.2. Indirection Cells

In order to cope with references with weight one, we make use of indirection cells. An indirection cell is a small object consisting of a single reference of weight one. Since the weight of the reference in an indirection cell is always one, its value need not be stored. When we wish to duplicate a reference with weight one we create an indirection cell containing a copy of the reference to be duplicated and having a maximum reference count. This indirection cell can be created on the same node as the object containing the reference to be duplicated, so no communication is necessary. Note that the reference count of the referenced object does not need to be changed. The reference to be duplicated is replaced with a reference to the indirection cell with maximum weight. This new reference can then be duplicated as normal (see figure 4.2-1).



Duplication of a reference with weight one using an indirection cell

Figure 4.2-1

4.3. Further Issues and Analysis of the Algorithm.

The paper [Bevan 1987] gives many more details and also shows how space overheads can be decreased by coding the refence information.

There are two ways in by which the performance of the algorithm can be measured, namely how many indirection cells an evaluator will have to go through in order to reach the object being referenced, and what the space overheads of recording the reference information are. Clearly the number of indirection cells is determined by how big the reference count and reference weight fields are. By taking into account the characteristics of the execution of functional programs, it is shown in [Bevan 1987] that the overheads are encouragingly small, typically between one and three bits per reference field for very good performance.

5. Conclusions and Further Work.

Two major breakthroughs have been made in this part of the project, namely the definition of a natural parallel reduction model and the discovery of a distributed reference counting garbage collection algorithm which has low overheads.

Using experience gained in looking at various parallel architectures for parallel graph reduction, and especially COBWEB, we have defined a distributed memory architecture which supports our parallel reduction model [Bevan et al 1987]. An emulation of our parallel reduction model could quite well be helpful in helping us understand how such a machine will work, and we need to now simulate our architecture.

6. Acknowledgements.

Our work has not proceeded in isolation, but has benefited greatly from many discussions with various people. We would like to single out especially Chris Hankin and Samson Abramsky of Imperial College London, Simon Peyton Jones of University College London, Peter Osmon of The City University and Malcolm Shute of Middlesex Polytechnic. The work on the distributed garbage collection algorithm would not have been done if we had not been forced to look more carefully at memory management by the Working Group on Architectures and Applications within ESPRIT project 415.

This work has been partially funded by ESPRIT Project 415 - Parallel Architectures and Languages for AIP : A VLSI-Directed Approach.

7. References.

[Anderson et al 1987]

Anderson, P., Hankin, C., Kelly, P., Osmon, P., and Shute, M., COBWEB-2 : Structured Specification of A Wafer-Scale Supercomputer, *PARLE (Parallel Architectures and Languages Europe)*, (Vol. 1), Eindhoven, The Netherlands, 15-19 June, 1987, Springer-Verlag LNCS 258, pp. 51-67.

[Aubusson and Catt 1978]

Aubusson, R.C., and Catt, I., Wafer Scale Integration - A Fault-Tolerant Procedure, *IEEE Journal of Solid State Circuits, Sc-13*, 3, 1978.

[Bevan 1987]

Bevan, D.I., Distributed Garbage Collection Using Reference Counting, Best Paper Award, *PARLE (Parallel Architectures and Languages Europe)*, (Vol. II), Eindhoven, The Netherlands, 15-19 June, 1987, Springer-Verlag LNCS 259, pp. 176-187.

[Bevan et al 1987]

Bevan, D.I., Burn, G.L., Karia, R.J., and Robson, J.D., Design Principles of a Distributed Memory Architecture for Parallel Graph Reduction, Submitted to : *Draft Manuscript*, January, 1987.

[Burn 1987a]

Burn, G.L., *Abstract Interpretation and the Parallel Evaluation of Functional Languages*, PhD Thesis, Department of Computing, Imperial College of Science and Technology, University of London, 1987.

[Burn 1987b]

Burn, G.L., Evaluation Transformers - A Model for the Parallel Evalution of Functional Languages (Extended Abstract), To be published at: *Third International Conference on Functional Programming Languages and Computer Architecture*, Portland, Oregon, September 1987.

[Burn, Hankin and Abramsky 1986]

Burn, G.L., Hankin, C.L., and Abramsky, S., Strictness Analysis for Higher-Order Functions, *Science of Computer Programming*, 7, November 1986, pp.249-278.

[Clack and Peyton Jones 1986]

Clack, C., and Peyton Jones, S.L., The Four-Stroke Reduction Engine, *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, Cambridge, Massachusetts, 4-6 August, 1986, pp. 220-232.

[Dijkstra 1980]

Dijkstra, E.W., *A Mild Variant of Combinatory Logic, EWD735, 1980.*

[Hankin, Burn and Peyton Jones, 1986]

Hankin, C.L., Burn, G.L., and Peyton Jones, S.L., A Safe Approach to Parallel Combinator Reduction (Extended Abstract), *Proceedings ESOP 86 (European Symposium on Programming)*, Saarbrucken, Federal Republic of Germany, March 1986, Robinet, B., and Wilhelm, R. (eds.), Springer-Verlag LNCS 213, pp. 99-110.

[Hankin, Burn and Peyton Jones, 1987]

Hankin, C.L., Burn, G.L., and Peyton Jones, S.L., A Safe Approach to Parallel Combinator Reduction, To be published in, *Theoretical Computer Science*.

[Hankin, Osmon and Shute 1985]

Hankin, C.L., Osmon, P.E., and Shute, M.J., COBWEB: A Combinator Reduction Architecture, in : *Proceedings of IFIP International Conference on Functional Programming Languages and Computer Architecture*, Nancy, France, 16-19 September, 1985, Jouannaud, J.-P. (ed.), Springer-Verlag LNCS 201, pp. 99-112.

[Hudak 1984]

Hudak P. and Goldberg B., Experiments in Diffused Combinator Reduction, *ACM Symposium on Lisp and Functional Programming*, Austin, Texas, USA, August 1984, pp 167-176.

[Hudak 1985]

Hudak P. and Goldberg, B., Distributed Execution of Functional Programs using Serial Combinators, *IEEE Transactions on Computers, Vol C-34* 10, October 1985.

[Johnsson 1987]

Johnsson, T., *Compiling Lazy Functional Languages*, PhD Thesis, Department of Computer Sciences, Chalmers University of Technology, 1987.

[Karia 1987]

Karia, R.J., *An Investigation of Combinator Reduction on Multiprocessor Architectures*, PhD Thesis, University of London, January 1987.

[Kennaway and Sleep 1986]

Kennaway, R., and Sleep, R., *Director Strings as Combinators*, University of East Anglia Technical Report, November 1986.

[Lerman and Maurer 1986]

Lerman, C.-W. and Maurer, D., A Protocol for Distributed Reference Counting, *Proceedings 1986 ACM Conference on Lisp and Functional Programming*, Cambridge, Massachusetts, August 4-6, 1986.

[Mycroft 1981]

Mycroft, A., *Abstract Interpretation and Optimising Transformations for Applicative Programs*, PhD. Thesis, University of Edinburgh, 1981.

[Peyton Jones 1987]

Peyton Jones, S.L., *The Implementation of Functional Programming Languages*, Prentice-Hall International Series in Computer Science, 1987.

[Peyton Jones, Clack and Salkild 1985]

Peyton Jones, S.L., Clack, C. and Salkild, J., *GRIP - A Parallel Graph Reduction Machine*, Dept of Computer Science, University College, London, November 1985.

[Seitz 1985]

Seitz, C.L., The Cosmic Cube, *CACM 28, 1*, January 1985.

[Shute and Osmon 1985]

Shute, M.J., and Osmon, P.E., COBWEB - A reduction architecture, *International Workshop on Wafer-Scale Integration*, 10-12 July, 1985, Southampton University, United Kingdom.

[Turner 1979]

Turner, D.A., Another Algorithm For Bracket Abstraction, *The Journal of Symbolic Logic 44* 2, June 1979, pp. 267-270.

[Turner 1985]

Turner, D.A., Miranda: A non-strict functional language with polymorphic types, *Functional Programming Languages and Computer Architecture*, September 1985, Nancy, Jouannaud, J. P., (ed.), Springer-Verlag LNCS 201, pp. 1-16.

[Wadler 1987]

Wadler, P., *Strictness Analysis on Non-Flat Domains (by Abstract Interpretation over Finite Domains)*, in Abramsky, S., and Hankin, C., (eds), *Abstract Interpretation of Declarative Languages*, Ellis Horwood, 1987. (Originally distributed on the FP mailboard November, 1985.)

[Wadsworth 1971]

Wadsworth, C.P., *Semantics and Pragmatics of the Lambda Calculus (Chapter 4)*, PhD Thesis, University of Oxford, 1971.

Project No. 415

S E T H E O
A SEquential THEOremprover for first order logic

Stephan Bayerl, Reinhold Letz

Forschungsgruppe Künstliche Intelligenz
Institut für Informatik
Augustenstraße 46 / II
8000 München 2

Tel.: +49-89-521098 / Telex: tumue d 05-22854
csnet: <name> % tumki.uucp @ germany.csnet

Keywords

Automatic theorem proving, first order logic, preprocessing, Connection Method, Model Elimination.

INTRODUCTION

The main goal of the Project 415 F is the design and implementation of a parallel theorem prover for first order logic. As an intermediate result the sequential version of the system has already been attained which is the topic of this paper. The system contains the following characteristic features:

- the Connection Method as engine of deduction,
- strong modularisation of all parts to enable independent working and interchangeability,
- a powerful preprocessing part to reduce the complexity of the input formula without violating its validity status,
- a connection graph module which enables the system to work only on a restricted part of the search space (reducing search effort),

- fanning out of clauses to facilitate entrance at any literal during the proof process,
- reordering of clauses to reduce the amount of backtracking,
- compilation of clauses into system functions and machine code which allows more compact code during generation of new clause copies.
- polynomial unification incorporating occurs–check to guarantee soundness of the proof procedure (unlike almost all PROLOG systems),
- completeness ensured by consecutively depth–bounded search (in contrast to all PROLOG systems),
- lemma generation to reduce the number of inference steps in proves with similar subgoals.

The system is implemented in FRANZ LISP and C. All essential features will be discussed in detail and advantages will be demonstrated with different examples.

PRELIMINARIES

We assume the reader to be familiar with the basic notions of predicate logic as they are presented e.g. in [Bi82, Cha73, Ro79, Smu68]. We stick to standard terminology but let us just emphasize our usage of a few essential concepts. SETHEO accepts different logical notations. By default the formulae are assumed to be in Skolemized normal form. If the input formula is expressed in standard notation containing quantifiers and logical operators it is first normalized by a linear and structure–preserving transformation. For details consult [Ed85a]. Just by convention the decision is here to deal with the refutation of formulae which is sufficient for reasons of duality. Hence in the sequel all formulae are supposed to be written in *conjunctive* normal form.

Accordingly a formula – or **matrix** – is a set of **clauses** which are sets of **literals**.

A **literal** is a (negated) atomic formula, i.e. consists of a predicate symbol followed by terms as arguments.

A **literal occurrence** in a matrix F is a pair (L,c) – abbreviated by L_c – where the literal L occurs in the clause c.

A **connection** $\{L_c, K_d\}$ in a matrix F is a two–element set of literal occurrences with $L \in c$ and $K \in d$ for clauses c, d in F, where L and K have same predicate symbols and different signs.

A connection $\{L_c, K_d\}$ is **weakly unifiable** if there are substitutions σ, τ of terms for variables such that $\sigma(L) = \neg\tau(K)$.

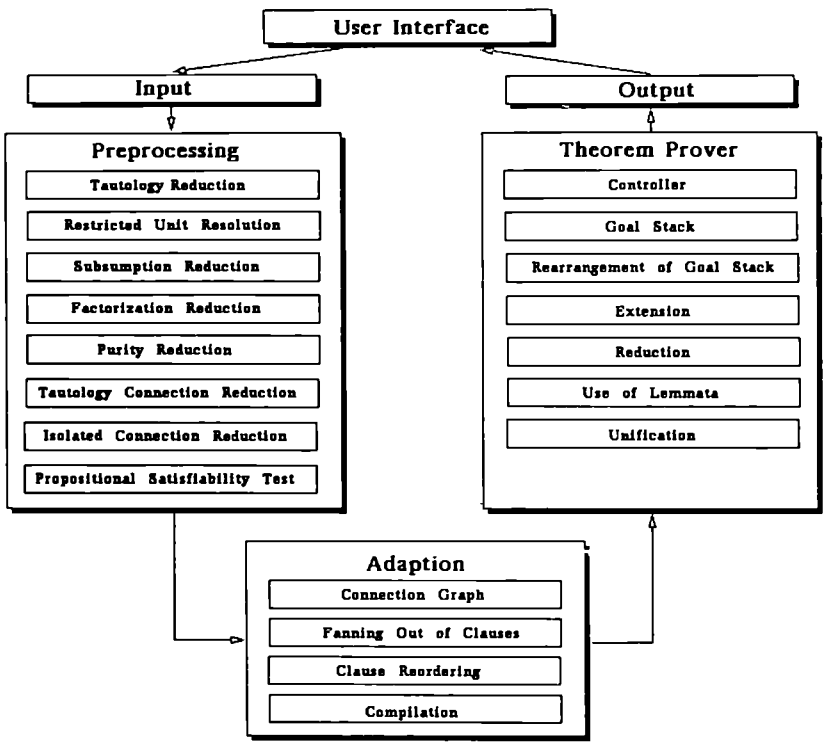
A **resolvent** **R**(C) with respect to a weakly unifiable connection $C=\{L_c, K_d\}$ is the union of $\sigma(c)\setminus\{\sigma(L)\}$ and a variable–disjoint renaming of $\tau(c)\setminus\{\tau(K)\}$, where σ and τ are most general substitutions induced by C (for details consult [Ro65]).

MODULARISATION

SETHEO is designed as a set of main modules.

- INPUT
- PREPROCESSING
- ADAPTATION
- MAIN PROOF MECHANISM
- OUTPUT

Each of these modules is devided into a set of smaller modules as displayed in the following graphic.



Picture 1: System Design

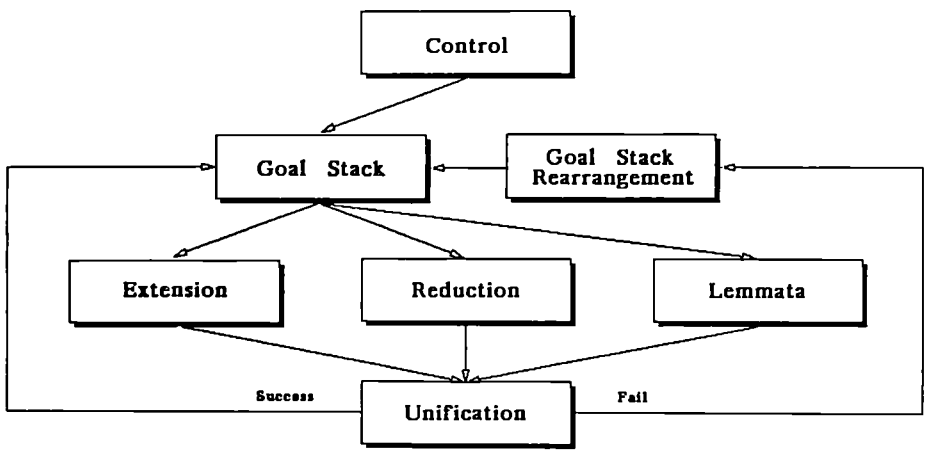
The **Input Module** takes a first order formula written in one of the notations given above, normalizes it, if necessary, and transforms it into an internal representation acceptable for the preprocessing part.

The **Preprocessing Module** includes complexity reductions as well as analysis of the propositional structure of the input formula.

For reasons of efficiency in the **Adaptation Module** the clauses – the output of the preprocessing – are fanned out, reordered and compiled into LISP functions.

The **Main Proof Mechanism** is a theorem prover for full first order logic based on a tree–structured specialization of the Connection Method [Bi82], which is as well known as the Model Elimination calculus [Lo78, Sti84]. The prover is a backtracking–driven search algorithm incorporating factorization. Completeness is guaranteed by limiting the number of inferences (or the depth of the search tree) and by successively increasing this limit.

The interaction between the several submodules of the proof procedure may be visualized with the help of a diagram:



Picture 2: Architecture of the Theorem Prover

Some of these submodules which are of greater interest for the understanding of the efficieny of the whole system will be discussed in separate sections (e.g. use of lemmata, connection graph, unification).

The **Output Module** gives the user information on the result of the proof as well as optional information on the process itself, e.g. trace, valid parts of the given formula or even the whole proof tree, which is shown as a two–dimensional graphic displaying all substitutions and connections between the respective literals.

PREPROCESSING

In the preprocessing part the complexity of the input formula is reduced with respect to the search space induced by the proof procedure. The performed modifications comprise reductions which preserve (un)satisfiability and in some cases even equivalence. Also the propositional structure of the formula will be tested for satisfiability.

The preprocessing module consists of eight independent submodules.

Equivalence Preserving

- Tautology Reduction
- Restricted Unit Resolution
- Subsumption Reduction
- Factorization Reduction

(Un)satisfiability Preserving

- Purity Reduction
- Tautological Connection Reduction
- Isolated Connection Reduction

Satisfiability Test

- Propositional–Satisfiability Test

The first four features are local processes, i.e. they may work correctly on any subclass of the formula. For the other ones it is essential to consider the formula as a whole.

TAUTOLOGY REDUCTION

If a clause c of the input formula contains a literal together with its negation, c cannot contribute to a proof of the formula. Hence it may be deleted.

RESTRICTED UNIT RESOLUTION

If there exists an *unit clause* c and a clause d, literals $L \in c$ and $L' \in d$ with complementary predicate symbols and an unifier σ for the arguments of L, L' such that no substituted variable of L' is contained in other literals of d, then the clause d is replaced by $d \setminus \{L'\}$.

SUBSUMPTION REDUCTION

According to the definition in [Ro65] we say that a clause c **subsumes** another clause d iff there is a substitution σ such that $\sigma(c) \subset d$.
All subsumed clauses of a formula can be canceled without changing the provability

properties of the formula. We have divided the process of removing subsumed clauses into two parts. At first we determine for each clause c the set of clauses $\hat{c} = \{c_1, c_2, \ldots, c_n\}$ with all predicate symbols of c occurring in $P(c_1) \cap P(c_2) \cap \ldots \cap P(c_n)$ where $P(c_i)$ is the set of predicate symbols of c_i.

This can be done easily with our representation of the input formula. Then we test subsumption of c with respect to the elements of $\hat{c}$. In general the result of a subsumption test between two clauses is negative. Therefore we do not use backtracking but implicit parallelism. Since the subsumption problem is NP-complete we employ in the case of large clauses a time limit to stop exponential explosion as a rather simple heuristics.

FACTORIZATION REDUCTION

If a clause c subsumes a proper subset d of itself c may be substituted by d.

PURITY REDUCTION

We call a literal L of a formula F **pure** in F, if there are no literals in other clauses of F that are unifiable with L. Clauses containing pure literals do not contribute to any refutation of a formula. Therefore they can be canceled out. By deleting some clauses from a formula new pure literals may emerge and initiate repetitive applications of this reduction process.

In checking a literal L for purity we are forced to find literals in other clauses that are unifiable with L. In this process previous information about the unifiability of connected literals is gained which can be used in the main proof procedure.

TAUTOLOGICAL CONNECTION REDUCTION

Let $C=\{L_c, K_d\}$ be a connection inducing most general substitutions σ, τ for the clauses c, d respectively. The connection C is called **tautological** iff either the resolvent R(C), $\sigma(c)$ or $\tau(d)$ are tautological clauses. In the Connection Method for any unsatisfiable formula there is a refutation without ignoring inferences on tautological connections.

ISOLATED CONNECTION REDUCTION

A literal L in a clause c of a matrix F is called **isolated** in F iff it is contained in just one (unifiable) connection $\{L_c, K_d\}$ of F and $c \neq d$. Notice that in the case where $c=d$ the literal L would be a pure literal in F. It holds that any matrix F containing an isolated literal in a connection $C=\{L_c, K_d\}$ is satisfiable iff $(F \setminus \{c\}) \cup R(C)$ is satisfiable. Thus an isolated literal may be eliminated by simply performing a resolution step and taking in the resolvent instead of the clause containing the isolated literal.

Obviously this procedure results in a proper reduction of complexity in case the partner of the respective literal is the member of a unit clause or, if both partners in the connection are isolated in which case we speak of an **isolated connection** [Bi87].

In any case an isolated literal can be specialized as induced by the respective connection. This may affect other literals in the matrix to become isolated as well resulting in a snowball effect.

PROPOSITIONAL–SATISFIABILITY TEST

This procedure tests the propositional structure of the formula for satisfiability. To this end we consider a formula where all terms of the initial formula are cancelled out. If the modified matrix is satisfiable, then this is also true of the unreduced input formula. In this case any further refutation attempt is useless.

Although all preprocessing modules are kept mutually independent, successful employment of one reduction rule may give rise to new applications of other ones, i.e. the reductions intensify each other. The investigation of powerful combinations and interactions between these modules opens a separate field of research which is not under discussion here.

ADAPTATION

CONNECTION GRAPH

Inspired by the concept of **weak unification** used in [Ed85b] an extended connection graph is employed. Ordinarily a connection graph expresses all possible connections between the literals occurring in a matrix neglecting unifiability of the respective literals. Hence in the execution of the proof procedure there is no information whether a unification fails *in principle* for two connected literals or because previous specializations are incompatible with the current one. Being able to distinguish these two cases may result in considerable reduction of the search space as is demonstrated with the formula

$$F_1 = \{ \{\neg P(x), \neg Q(f(x))\}, \{P(a_1)\}, \{P(a_2)\}, \ldots, \{P(a_n)\}, \{Q(g(a_i))\}, \{Q(f(a_n))\} \}.$$

A PROLOG–style top–down interpreter needs 3n unifications until success, whereas deletion of $Q(g(ai))$ from the possible alternatives for $\neg Q(f(x))$ reduces the number of unifications to 2n.

FANNING OUT

It is possible to view – similar to PROLOG – any clause with more than one literal as a *rule*. The literal at which a clause is entered plays the role of the momentary *head* and the remainder (*tail*) makes up the current *subgoals*. Clauses with only one literal are the *facts* and any clause could function as a *request*.

Instead of the complicated generation of subgoals depending on the current entry point during run–time we use a LISP function yielding all subgoals for a chosen head literal. This can be done by representing every clause with n literals by n new clauses with

different heads.

One advantage is the fast access of connected subgoals, another one the employment of heuristics during compile time, e.g. clause reordering.

CLAUSE REORDERING

An even more powerful method to reduce backtracking is supplied by the reordering of clauses. Recalling the previous example F_1 the indispensability of such a feature is obvious, as a reordering of $\neg P(x)$ and $\neg Q(f(x))$ decreases the amount of unifications from 3n to n+1.

In more detail we favour the following preference rules where the earlier ones are dominating the later ones:

- prefer subgoals with the smallest *weight* with respect to the matrix,
- prefer most specialized subgoals (e.g. ground literals),
- prefer subgoals that share most variables with the respective head.

The first preference rule says that subgoals with the smallest total value of connections will be pushed to the front of the subgoal list. Each connection starting from a subgoal gets a value depending on the number and the cardinality of the connected clauses. As a very simple strategy we calculate the weight of a literal occurrence as the sum of the cardinality of all connected clauses.

Example. The weight for the first subgoal in the rule *r*:

R(x) :- P(z), Q(z), ¬S(z).

in a formula containing the other clauses

```
?- R(x).
P(z) :- T(a), ¬S(z), ¬Q(z).
P(z) :- M(x), Q(f(a)).
P(z) :- R(z).
P(a) :- L(c), S(b), Q(x).
P(a).
P(b).
P(f(a)).
P(f(b)).
P(f(g(a))).
 . . .

Q(w) :- ¬S(w).
¬S(f(b)).
 . . .
```

is

number of (unifying facts)	5
number of (unifying rules)	4
number of (subgoals in all unifying rules)	+ 9
	18

By this heuristic the tail of any rule will be reordered prefering subgoals with the smallest weight. For instance the reordering of the rule *r* yields:

$$\mathbf{R(x) :- \neg S(z), Q(z), P(z).}$$

A solution using the bold formulae together with the reordered rule needs only 5 instead of 17 unifications. *End of example.*

The second preference rule says that the tail of a rule will be reordered according to the degree of specialisation of its subgoals, i.e. a constant ranges over a variable.

With the third one we take advantage of the fact that subgoals having many variables in common with the respective head are with a higher probability specialized than others as they may be instantiated by previous inferences affecting the head.

Application of these two heuristics is demonstrated with the clause

$$\neg P(x,y) :- \neg Q(z,w),\ R(a,b),\ R(z,w),\ R(w,a),\ \neg Q(y,x),\ S(a),\ Sf(a).$$

where the second rule dominates the third one. Reordering yields as result the succession

$$\neg P(x,y) :- R(a,b),\ Sf(a),\ S(a),\ \neg Q(y,x),\ R(w,a),\ \neg Q(z,w),\ R(z,w).$$

A similar approach may be found in [Bl81].

CLAUSE COMPILATION

In general a proof procedure, if based on the Connection Method or on any other calculus, requires copying of variables. Using structure sharing in LISP, copying can be very time consuming, as information about objects already created has to be gathered and consulted every time a variable, constant, or term is processed. During a compilation phase in advance of the main proof process the information about structure sharing is gathered only once and stored statically in the compiled code.

The advantages of this decision are numerous. The amount of garbage data produced can be reduced substantially, since no sets of *terms visited* have to be dynamically created each time a copy of a clause is made. Variables occurring only once in a clause need no overhead in the copy procedure. The resulting functions can be compiled into fast machine code. Finally the concept may be transferred easily to programming languages without names.

THEOREM PROVER

MODEL ELIMINATION

The *procedure* of finding a refutation in the Connection Method can be easily described with a tree or tableau instead of a static display of the respective connections in a matrix. Model Elimination is a specialization of the Connection Method based on this idea. It is an analytic calculus based on semantic trees in the tradition of the tableaux calculus [And81, Smu68].

Definition. A Model Elimination refutation or **tableau** T of a formula F is a pair (t,μ), where t is a finite tree and μ is a function from the non-root nodes of t onto literals satisfying the following conditions:

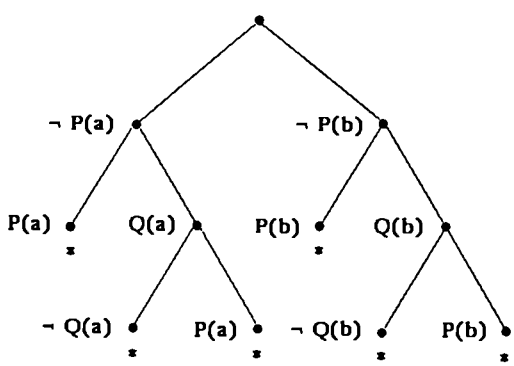
- for any equivalence class N of the *brother* relation on the non-root nodes of t – i.e. for the set of nodes with the same immediate predecessor – the range of N under μ is an instance of a clause of F.
- any leaf L of t has a predecessor K such that $\mu(K)$ contradicts to $\mu(L)$ – in this case L and K are called **complementary**,
- any other non-root node has at least one complementary node among its immediate successors. *End of Definition.*

It is useful to differentiate the single construction steps of such a tree starting from the immediate successors of the root (the **start clause**) into **extension**- and **reduction**-steps.

Extension step: take an open end l of the momentary tree and attach as immediate successors of l a clause instance of the formula containing $\neg$l, which is determined to be a leaf of the final tree.

Reduction step: an open end of the momentary tree, which has a contradicting predecessor is made a leaf of the final tree.

Consider the formula $F_2=\{\{\neg P(a), P(b)\}, \{P(x), Q(x)\}, \{\neg Q(x), P(x)\}\}$ and a tableau demonstrating the unsatisfiability of F_1. A star indicates the *complementarity* of a path.

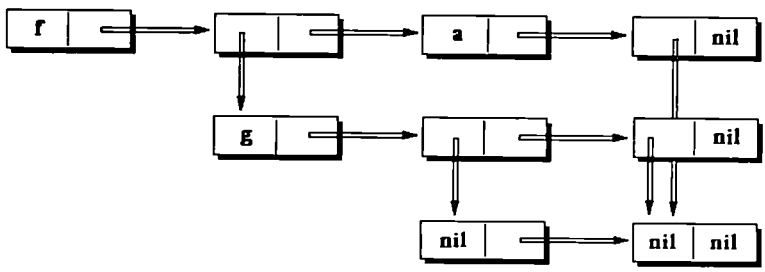


Picture 3: Tableau for $F_2=\{\{\neg P(a), P(b)\}, \{P(x), Q(x)\}, \{\neg Q(x), P(x)\}\}$

INFERENCING AND DATA STRUCTURE

Terms are internally represented as lists with constants and function symbols as LISP atoms. Thereby we follow the LISP convention that a function symbol is pushed in and is taken as the CAR of a list while its argument list makes up the CDR. Thus the term f(a,g(b,c)) is represented as the construct (f a (g b c))).

The treatment of variables is somewhat special. They are not represented as LISP atoms but as nameless CONS-cells, where identity of variables means that the respective pointers address just the same cell. This is advisable for an efficient working with destructive procedures such as the reorientation of pointers, which we use in unification. Consider the following picture as an illustration of the term f(g(x,y),a,y), which is the LISP expression: (f (g (nil) (nil)) a (nil)).



Picture 4: Data Structure in Pointer Notation

With the decision to use such a type of structure sharing two instances of a clause are totally equal in structure but different in addresses. The solution of the problem of generating clause copies is now easily solved by defining a LISP function that constructs a new instance of a clause any time it is needed. The clauses are accessed via the extended connection graph, in which for every literal of the matrix the names of the clause functions with the possible partners as head literals are hold. In this implementation the predicate symbols of the literals are even replaced by the names of the respective clause functions. Performing an inference step means calling a clause function pushing its value onto the stack in place of the goal in question. If unification with the head succeeds head and goal are removed and in case there are further alternatives available for the goal a backtrack point is established. Then the process is repeated with the first subgoal of the entered clause which is now on top of the stack.

UNIFICATION

The unification algorithm is due to [Co83]. This is essentially the Robinson algorithm, but by sharing the structure of terms of any size it is pushed down to quadratic complexity. In our implementation we do not use backward pointers to enable the reorientation of pointers but reference cells, which are introduced with every single step in a unification. It should be emphasized that no additional data structure is needed to support unification – the main reason for choosing this algorithm.

To minimize the performing of an occurs–check within a unification the simple heuristic is used that it may be omitted in extension steps into clauses where no variable occurs more than once inside the head [Pla84].

DEPTH–BOUNDED SEARCH

Completeness is guaranteed by limiting alternatively

- the number of inferences
- the depth of the proof tree

and by increasing this limit successively.

In choosing the first completeness mode with each successful inference step the number of available inferences is diminished by the cardinality of the entered clause. This mode is very successful in case the resulting *proof tree is not well–balanced in depth.* Furthermore this strategy guarantees that the proof tree with the smallest number of inferences will be found. On the other hand it is not very reasonable to increase the number of inferences only by one for each proof attempt.

Favouring the other alternative is strongly recommendable in a situation where a *relatively even solution tree is to be expected.* Unfortunately with this approach the number of inferences is difficult to control as it depends on the possible branching of the tree.

USE OF LEMMATA

The use of lemmata more or less corresponds to factorization in Resolution. It may reduce the number of inferences considerably, if one and the same literal occurs multiply as subgoal in the proof tree. In the case of *Horn clause reasoning* the incorporation of this feature is relatively unproblematic for reasons of completeness of Input Resolution [Lo78] which means in our terminology that no reduction steps are necessary. As a consequence in Horn clause reasoning ordinarily only extension steps are performed. Hence a lemma has the logical status of a unit clause and may be added to the list of facts of the respective formula. For illustration consider the PROLOG program for Fibonacci numbers

```
fib(0,1).
fib(1,1).
fib(N,F):- N2 is N-2, N1 is N-1,
           fib(N2,F2), fib(N1,F1), F is F2+F1.
?- fib(n,F).
```

where n stands for any natural number. The program obviously induces proof trees with an exponential number of nodes with respect to n. More precisely in the computation of Fibonacci(n) we need

$$\sum_{i=0}^{n} \text{Fibonacci}(n-i)$$

inferences involving fib-predicates. Thus e.g. for n=9 the subgoal fib(2,F) has to be solved 21 times in which case it makes sense to include fib(2,2) as an additional fact.

In the transition to *full predicate logic* the lemma feature may be generalized applying as well to subgoal solutions containing reduction steps. But in this case lemmata do not have the status of unit clauses in general. If the solution of a subgoal depends on reduction steps using predecessors of this subgoal, the generated lemma is a rule with the respective subgoal as head and the employed predecessors as subgoals. As the lemma feature is no necessary condition to guarantee completeness we decided for the present implementation to take into account only lemmata without reference to predecessors.

In any case the generation of lemmata during run-time is a very time-consuming process and should be carefully restricted to exceptional cases only. Momentarily we apply this feature only, if two literals in a clause have a common specialization.

PERFORMANCE

In the current FRANZ LISP implementation the main proof procedure runs with machine-coded clause functions appr. 2.7 K lips on a TARGON 35.

OUTLOOK

The system is designed to be generalized easily to a parallel theorem prover. Particularly its modularity facilitates independency in execution. Additionally we aim on *different layers* of parallelism

- competitive (i.e. pursuing different strategies),
- module extern (i.e. simultaneous performance of several modules),
- module intern (i.e. concurrent execution of module functions),
- function intern (e.g. in unification or subgoal selection).

These aspects will be integrated into the highly PARallel THEOrem prover PARTHEO which will be tested on different existing architectures to achieve the optimal hardware for its execution.

ACKNOWLEDGEMENTS

We would like to thank Wolfgang Bibel for valuable comments and Norbert Trapp for helpful discussions.

This work has been supported by the EC and by Nixdorf Computer AG within ESPRIT project 415 "Parallel Architectures and Languages for A.I.P.".

REFERENCES

[And81] Andrews, P.B.; Theorem Proving via General Matings; JACM Vol 28 Nr 2, pp 193–214 (1981).

[Bay86] Bayerl, S. & Kurfess, F. & Letz, R. & Schumann, J.; PROTHEO/2: Sequential PROLOG–like Theorem Prover based on the Connection Method; ESPRIT 415F Deliverable D5 (1986).

[Bi82] Bibel, W.; Automated Theorem Proving; Vieweg Braunschweig (1982).

[Bi83] Bibel, W.; Matings in Matrices; CACM 26, pp 844–852 (1983).

[Bi87] Bibel, W. & Letz, R. & Schumann, J.; Bottom–up Enhancements of Deductive Systems; Conference on Artificial Intelligence and Information–Control Systems of Robots, North–Holland, Amsterdam (to appear 1987).

[Bl81] Bläsius, K. & Eisinger, N. & Siekmann, J. & Smolka, G. & Herold, A. & Walther, C.; The Markgraf Karl Refutation Procedure; Proceedings of the 7th IJCAI Vol 1, pp 511–518 (1981).

[Cha73] Chang, C.–L. & Lee, R.C.–T.; Symbolic Logic and Mechanical Theorem Proving; Orlando et al. (1973).

[Cor83] Corbin, J. & Bidoit, M.; A Rehabilitation of Robinson's Unification Algorithm; Information Processing, North–Holland, Amsterdam (1983).

[Ed85a] Eder, E.; Properties of Substitutions and Unifications; JSC Vol 1 (1985).

[Ed85b] Eder, E.; An Implementation of a Theorem Prover Based on the Connection method; Art. Intell. (W. Bibel, B. Petkoff, eds.); North–Holland, Amsterdam, pp 121–128 (1985).

[Lo78] Loveland, D.W.; Automated Theorem Proving: a Logical Basis; North–Holland, Amsterdam (1978).

[Pla84] Plaisted, D.A.; The Occur–check Problem in Prolog; New Generation Computing, Vol 2 Nr 4, pp 309–322 (1984).

[Ro65] Robinson, J.A.; A Machine Oriented Logic Based on the Resolution Principle; JACM, Vol 12 Nr 1, pp 23–41 (1965).

[Ro79] Robinson, J.A.; Logic: Form and Function; North-Holland; New York (1979).

[Smu68] Smullyan, R.M.; First Order Logic, Springer, Berlin (1968).

[Sti84] Stickel, M.E.; A Prolog Technology Theorem Prover; New Generation Computing, Vol 2 Nr 4, pp 371–383 (1984).

Project No. 415

MULTI-LEVEL SIMULATOR FOR VLSI
- an overview - *

P. Mehring and E. Aposporidis

AEG Aktiengesellschaft, Berlin Research Institute
Hollaenderstrasse 31-34, D-1000 Berlin 51

Abstract

Simulation is a key element in modern and future digital circuit design. However, simulation becomes a bottleneck with increasing design complexity. There are mainly two ways to get out of this situation: reduction of the simulation load through multi-level simulation and acceleration of the simulation through exploitation of parallelism.

This paper deals with the development of a VLSI-Simulator which combines both approaches to achieve optimal performance. It is an informal overview of the work of AEG and its subcontractor Technische Universitaet Berlin carried out within ESPRIT Project 415.

1. INTRODUCTION

Within the frame work of ESPRIT 415, Philips and AEG are investigating the object-oriented approach in a joint venture. The object-oriented approach is considered as a natural evolution from current programming styles.

AEG's task is to develop one of the three demonstrators for the machine, namely a multi-level VLSI-simulator, together with its subcontractor Technische Universitaet Berlin. The demonstrators are being developed in parallel with the architecture for ensuring optimal design.

Powerful simulators are a key element in modern and future digital circuit design environments. However, simulation becomes a bottleneck in the circuit design with increasing design complexity.

There are mainly two ways to get out of this situation:

- Reduction of the simulation load through so-called multi-level simulation, i.e. using different modelling levels within a circuit and not only a basic level throughout, e.g. gate level or electrical level.

- Acceleration of the simulation through exploitation of parallelism on the different modelling levels and simulator levels.

The multi-level VLSI-simulator combines both approaches to achieve optimal performance.

---

* Research partially funded by ESPRIT Project 415: Parallel Architectures and Languages for Advanced Information Processing - A VLSI-Directed Approach.

2. LEVELS OF ABSTRACTION IN THE DESIGN PROCESS

VLSI-circuits are very complex systems and can not be designed in one single step even with modern computer aided design techniques. There are four major design steps as shown in Figure 1.

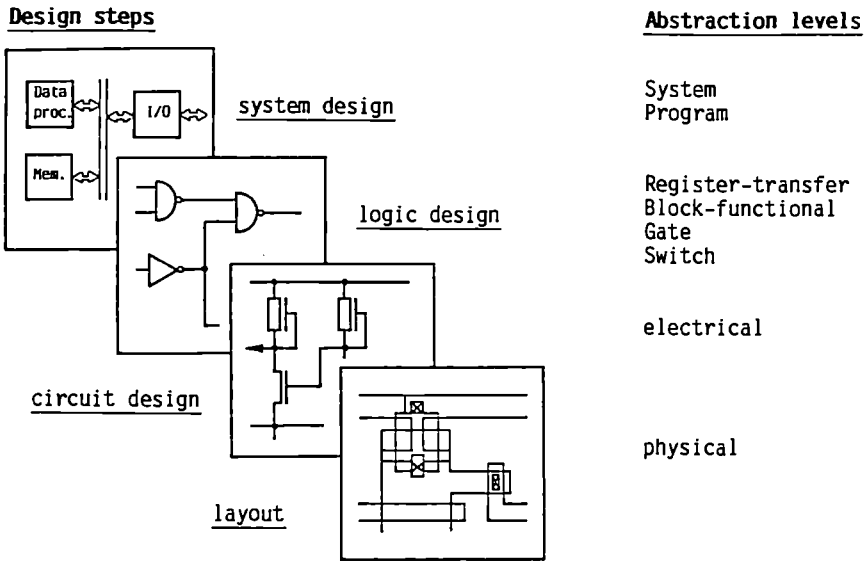


Figure 1: Steps in the circuit design [2]

The design starts with a global specification which is formulated at the highest possible level of abstraction and covers all system requirements. Then the detailed structure of the system is derived by breaking down this abstraction level step by step. In this process the designer tries to break down the problem into a number of interconnected subproblems. This process is repeated until solutions to all the subproblems are known or until well-known procedures are available which can be applied to solve these subproblems (top-down design).

At least eight levels of abstraction have become recognized (see Figure 1). Each abstraction level has a related notation for describing the system (i.e. the components, ways of combination and rules of behaviour).

In the following we focus on the logic design levels as these are the most important levels for multi-level simulation.

3. PRINCIPLES OF DIGITAL-CIRCUIT SIMULATION

Simulation methods are characterized by the circuit model used and by the procedures for model handling. Models may represent different levels of abstraction corresponding to the levels in Figure 1. There is a large variety of models. Each model is characterized by its modelling of signals and its modelling of elements.

There are two types of simulation in logic design corresponding to different phases of the design including production:

- *logic simulation* for checking the design for logic and timing errors
- *fault simulation* as an aid in testing the circuit for possible hardware defects.

3.1 Simulation models

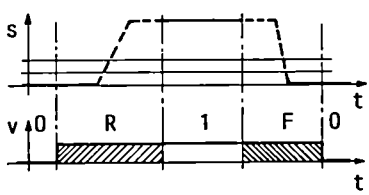
Modelling of signals

At the *logic design levels* (switch, gate, block-functional and register-transfer) the analog signals are replaced by discrete approximations [5, 4].

As a first approximation the stationary (binary) values 0 and 1 are used. In order to mimic the dynamic behaviour of the elements more closely, the signal transitions must be introduced into the model. At least one additional value is needed, e.g. U for the unknown state. The modelling of edge triggered elements, however, requires the introduction of two additional values - R (Rise) for a transition from 0 to 1 and F (Fall) for a transition from 1 to 0 (see Fig. 2).

Apart from these values, which serve to mimic the underlying analog signal, additional values or states are required to model particular technology or circuit specific characteristics. The following are commonly used:

- to model the high impedance condition of an element output (in general this refers to gate elements used in connection with busses) the additional value Z is used.
- to model MOS transistors as bidirectional switches at the "switch level" strength classes are added. They specify the type of connection between the voltage source and the node.



plus: U, X, Z,
strength classes

Figure 2: Modelling of circuit signals for a stochastic signal transition
s=signal, v=variable

Normally, these additional values are only used in those parts of the circuit where they apply (i.e. transmission gates, wired outputs and bus connections).

Modelling of elements

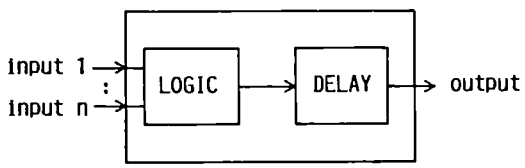
Modelling of circuit elements splits down into modelling of their logical behaviour and of their timing behaviour.

As an example we focus on the gate level, the basic level for the logic design. At this level the elements are gates, i.e. unidirectional basic elements with boolean logical functions.

In order to simplify the modelling of elements, the logical function is separated from the delay characteristics of the elements. Delays are represented by a delay element (DELAY) which is placed in series with the logic element (LOGIC) (Fig. 3).

The logic operation can be performed by an algorithm corresponding to the gate type or by a truth table (table look up-technique). As an example, the truth table of a NAND-gate for a tri-valued signal is shown in Figure 3.

The propagation *delay* of the circuit elements causes a delayed output reaction to a change at the input.



-1

A \ B	1	0	U
1	0	1	U
0	1	1	1
U	U	1	U

-2

Figure 3: Modelling of circuit elements
- 1 Model of a gate
- 2 Truth table of a NAND-gate for a tri-valued signal

3.2 Simulation execution control

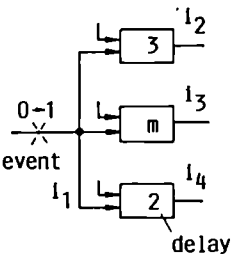
There are three basic methods for the control of the execution of a simulation: "compiled simulation", "event driven" (or "table driven") simulation and "interpretive simulation".

The most important execution control method is the *event-driven* (or *table-driven*) simulation. This method is applied in by far the greatest part of available simulators for digital circuits.

With this method "element calculations" are triggered by "events", i.e. changes of signal values. The "simulation time" is not incremented by fixed units of time but jumps from one event instant to the next. Moreover, only those elements are (re)-calculated which are immediately affected by an event, i.e. those of which at least one input signal has changed (selective trace).

An "event list" listing all future events (input-events as well as evaluated events) is used to keep track of the events.

The simulator fetches from the event list an event from the next time instant, enters the new variable values into the corresponding list and determines via the "connection list" those elements which are activated by this event. These elements are then calculated in sequence and resulting new events are entered into the event list (Fig. 4).



event list

t	t+2	t+3	...	t+m
i_1	i_4	i_2	..	i_3
:	:	:	..	:

Figure 4: Principle of event-driven simulation.
The event i_1 propagates to three circuit elements (with delay of 3, m and 2 units of time)
The evaluation of these elements at simulation time t results in three new events (i_2, i_3, i_4) to be processed at simulation time t+3, t+m, t+2

3.3 Fault simulation

Fault simulation serves in checking the quality of test patterns developed for a digital circuit. To this end the digital circuit is simulated assuming certain defects with their binary effects being injected into the circuit model. The results at the outputs are compared with the fault free case (good circuit).

The core of every fault simulation is, therefore, logic simulation for all fault configurations. It is to be determined how many of the assumed defects are detected and how many are not, and ultimately a "fault catalogue" is to be produced containing the detected faults.

The simplest fault simulation method is *single fault* simulation, in which every fault is simulated in a separate simulation run. This method is inefficient. A much more powerful approach is the so-called *concurrent* method, in which the whole set of faults is simulated concurrently. The basic idea here is to use the simulation of the good circuit as a reference for the faulty circuits. The faulty circuits execute explicitly only the changes to the reference behaviour; the rest is "borrowed" from the reference simulation.

3.4 Multi-level Simulation

Traditionally the design and simulation of a circuit is carried out on just one level, the gate level, using elementary logic operations such as AND, OR, etc.. However, simulation of VLSI-circuits at this level only is not efficient.

Because of reasons of time and cost multi-level methods have to be applied; i.e. simulation is accomplished at different levels of detail for the various parts of a circuit. This allows the cost of simulation to be considerably lowered compared to gate-level simulations.

With multi-level simulation a digital circuit is first of all partitioned into subcircuits and the subcircuits are modelled on different abstraction levels. These models are then translated or compiled into the corresponding simulator lists.

4. EXPLOITATION OF PARALLELISM

The basic principle behind the organisation of parallel processing in the simulation of digital circuits is partitioning with regard to structure and time. There are three basic levels for the organisation of parallel processing plus special techniques on hardware level.

Level	Mechanisms
Case	Partitioning of: fault set, stimuli set, variant set
Circuit	Circuit partitioning, macro-pipelining, asynchron. proces.
Algorithm	Algorithm partitioning, task pipelining, event handling
Hardware	Implementation parallelism

4.1 Parallelism on the case level

On the case level, i.e. on the level of simulation jobs, there is a very important possibility for parallel processing for all of the types of variant simulation:

- the structure variants, such as with fault simulation and the simulation of design variants,
- the stimulus variants, such as, for example, with the simulation of instruction variants (e.g. addressing modes) of a processor.

The basis is that all the variants can be executed as single-case simulations independently from each other; i.e. in parallel.

Figure 5 illustrates the principle of variant simulation in the case of fault simulation. To this end the set of variants is divided into subsets which are then processed as parallel jobs.

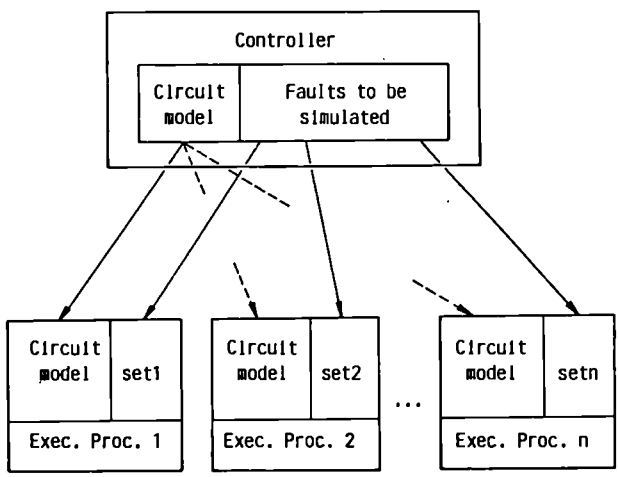


Figure 5: Principle of variant simulation in case of fault simulation

Because of cost reasons variant simulations have been done routinely only for fault simulation because fault simulation is indispensable for the analysis of testability and the preparation of test patterns.

Consistent use of VLSI technology and the cost savings which arise from this will also make the systematic testing of design variants economical.

4.2 Parallelism at the circuit level

At the circuit level, i.e. the level of individual circuit simulation, the basis for parallel processing is partitioning of the circuit into subcircuits. Modern design techniques (top down design) lend themselves automatically to a hierarchical partitioning of circuits into functional units.

On the highest partitioning level a combinatorial type of linking of functional units usually results. Such functional units can be treated independently from each other, i.e. each can be ascribed a "virtual simulator".

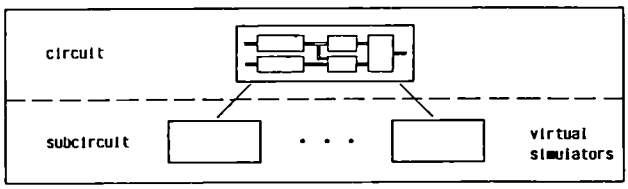


Figure 6 : Circuit partitioning

Circuit partitioning ascribing subcircuits to virtual simulators allows the exploitation, individually or in combination, of the following forms of parallel processing:

- Parallel processing of synchronous and asynchronous subcircuits between clock points,
- Macro-pipelining of the processing of subcircuits, and
- Asynchronous parallel processing of all subcircuits.

For asynchronous parallel processing of subcircuits there are two principally different strategies :

- Conflict-free strategy according to the principle of the so-called "delimited action" ("Worst case strategy") [8]
- Strategy with conflicts according to the principle of the so-called "time warp" ("optimistic strategy") [6].

As shown in [9], the "time warp" principle is well suited for the object-oriented machine. This principle is also providing the basis for current experiments at the Cosmic Cube [11] within the framework of discrete simulations [6, 12].

4.3 Parallel processing at the algorithm level

The algorithm level, i.e. the level of individual virtual simulation machines, offers very important and distinct options for parallel processing. It is the major level for exploitation of parallelism in today's "simulation machines".

In the following we concentrate on the event-driven algorithm.

Figure 7 illustrates the principle sequence of operations in the event-driven algorithm and the inherent parallelism [1]. The "signals" are concurrently propagated along the different paths (vertical lines).

Conceptually all the operations on the same horizontal level may be performed in parallel. Note that different concurrent events may be propagated to the same element which may lead to a multiple input change.

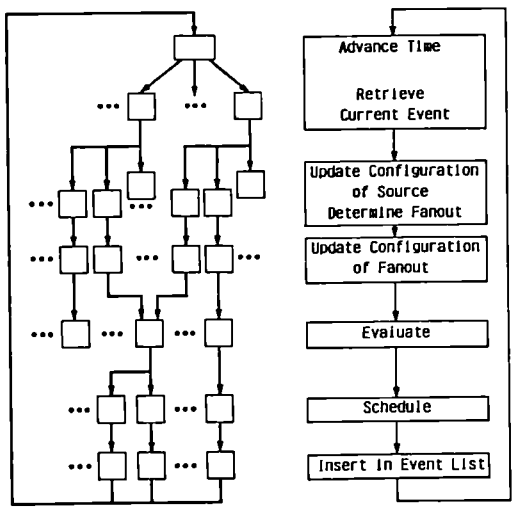


Figure 7 : Concurrency in Logic Simulation (event-driven algorithm) [1]
- inherent parallelism
- principle sequence of operations

A very efficient and common approach is to map the sequence of operations onto a circular pipeline with a central event list.

4.4 Parallel processing at the hardware level

At the hardware level there is a whole bunch of options for exploitation of parallelism, e.g. through processor tailoring, parallel communication path, use of parallel table look up techniques for the evaluation of elements, data caching etc..

These are implementation details which heavily depend on the choice of methods and machine architecture. Both points are still under study.

5. PRINCIPLE FEATURES OF THE SIMULATOR

The multi-level simulator covers the logic-design cycle, i.e. from register transfer level to functional level, gate level down to switch level with provisions for including the programming level and the electrical level and support for hardware-in-the-loop simulations. We expect that the lower level design cycles, namely the electrical and physical design cycle, will be largely automated in the near future.

The architecture exploits the inherent parallelism by use of the following principle techniques (Fig. 8):

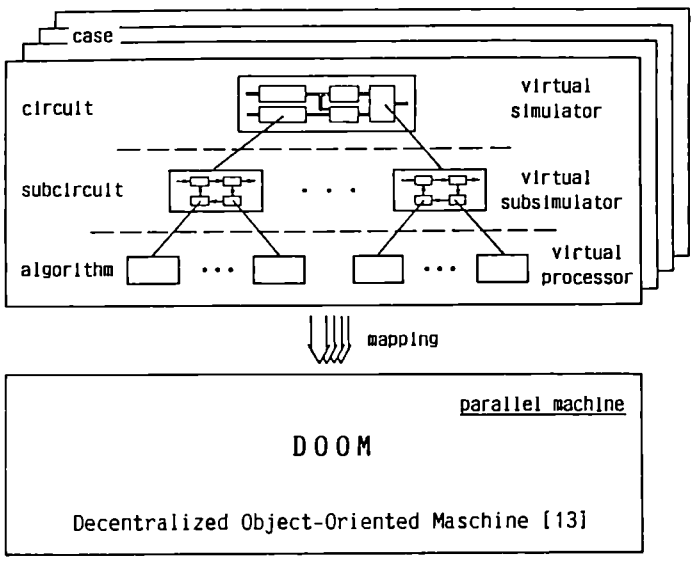


Figure 8 : Exploitation of parallelism

- Case partitioning with multi-case simulations, i.e. simulation of structure variants such as in fault simulation, design variants, stimuli variants.

- Functional circuit partitioning, i.e. partitioning of a whole circuit into subcircuits (functional units). Processing of subcircuits will be as far as possible asynchronous, e.g. using the so-called time-warp method.

- Algorithm partitioning primarily into pipelined tasks. How far multiple pipelines or a totally parallel approach are possible is yet to be determined.

The major question is how to match these techniques with the machine architecture for an optimal trade-off between node architecture and communication.

6. GOALS OF THE PROJECT

The fundamental goal in the development of the multi-level simulator is to gain precise insight into the different aspects of parallel machines in a sensible application, from architecture and programming to performance increase and cost effectiveness.

More detailed goals are:

- drastic simulation performance increase by use of multi-level simulation and massive parallel processing.

  We strive for a speed-up by a factor of more than 100 in logic simulation and a near linear speed-up in fault simulation on a 1000-node machine through parallel processing techniques. Additional speed will come from an advanced compute engine within the nodes by exploiting VLSI-technology.

- Significant improvement of cost effectiveness through the use of VLSI technology for the simulation itself. The nodes will be of RISC-type.

- Substantial reduction of cost for software production and maintenance through use of the object-oriented programming style.

- Extensibility of architecture (especially with regard to maximum design complexity as well as new simulation techniques, e.g. symbolic processing or knowledge based components).

The goal is an open simulation architecture and a prototype implementation.

7. PRESENT STATUS AND RESULTS

The main activities and results in the first half of the project, are:

- Development of a simplified experimental simulator (SILKE) and measurements with SILKE on a number of machines.

- First implementation of the experimental simulator in the parallel object-oriented language POOL on the POOL-simulator (SODOM).

- Specification of SILKE in FP2.

- Study of the above mentioned fundamental techniques of exploiting parallelism with prototype implementations [3].

7.1 Experimental Simulator

The experimental simulator is intended to serve as a vehicle for testing ideas, mechanisms and machines. It was originally written in PASCAL and later on rewritten in POOL, OCCAM and C. The basic version contains only the gate level; the first multi-level version includes the register transfer level additionally. The implementation of the switch level as well as parallel versions (see later "circuit partitioning") are underway.

The SILKE performance was measured on a number of complex instruction set machines (VAXes, 68020, 80286) and RISC-type machines (TRANSPUTER, Fairchild CLIPPER). The results support our option for a RISC-type compute engine.

7.2 Implementations in POOL

For checking the suitability of the object-oriented language (POOL-T) for parallel logic simulation and the language constructs needed, three different experimental approaches have been exercised:

- the *Circular Pipeline* as the "classical" approach for parallel processing at the algorithm level. This approach is similar to the pipeline in [1], see Fig. 7; it uses a central event list.

- in the "natural" *Gate/Object* approach each gate was modelled by one object. Objects communicate by sending event messages. An event is the tuple (value, time stamp). Each input of a gate is modelled by a FIFO-queue through which the time-ordered stream of event messages flows, i.e., each gate has his own event list.

  This approach is the most natural one for object-oriented simulation and can be regarded as the extreme case of circuit partitioning, i.e., down to the gate level.

- the *Circuit Partitioning* approach using *Time Warp* synchronization performs parallel processing at the circuit level (see section 4.2, Fig. 6). This approach (which uses distributed event lists) is novel.

All these approaches implement the specification of SILKE. The underlying basic principles for simulation execution control are event-driven simulation with selective trace.

First measurements have indicated granularity problems. For solving the problems two different directions are performed: enhancements in the object-oriented language (POOL-2) and changes in the implementation of the simulator to achieve coarser granularity.

A detailed description of the implementations and the discussion of the results is given in [9].

7.3 Specification in FP2

In connection with work of the Working-Groups in the project, a formal specification of SILKE in FP2 [7] was done in collaboration with LIFIA, as reported in [10]. The specification runs on the FP2-interpreter.

This specification serves as a basis for "benchmarking" the various programming styles in project ESPRIT 415 (object-oriented, functional, logic, data-flow) using SILKE as application example.

7.4 Exploitation of parallelism

In the field of exploiting parallelism (see section 4.) the following results were achieved.

Case partitioning

We successfully implemented a method for dynamic distribution of a fault simulation task in a network. The implementation is based on a state-of-the-art fault simulator (DISIM fault simulator [4]), as shown in Fig. 9.

The main component of the "parallel fault simulator" is a SUPERVISOR which controls all "servers" and schedules them appropriately.

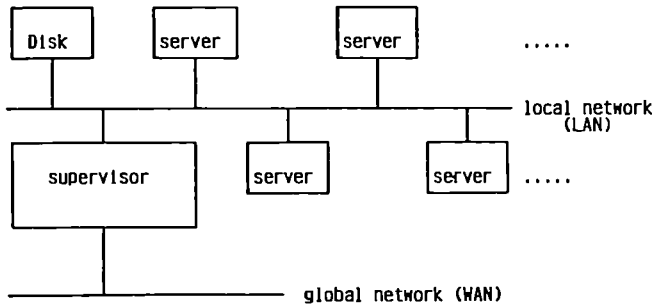


Figure 9: Initial implementation of the parallel fault simulator

To give the SUPERVISOR dynamic control over the fault simulations running on the servers, a special "fork on external request" was added to the DISIM fault simulator. This external request is mapped onto an existing internal mechanism in the fault simulator for splitting the actual fault set in case of memory overflow.

The SUPERVISOR uses this mechanism for *dynamically splitting* the fault sets in the servers - as shown in Figure 5 - so that a nearly optimal load balancing of servers is achieved.

Simulation results for networks with up to 64 nodes using a VAX-based network with four physical processors are shown in Figure 10. As expected, the speed up is nearly linear with the number of processors, as long as the ratio between fault simulation and good circuit simulation *per node* (w_n) is reasonable (in the order of 10).

The speed-up degradation for smaller ratios is due to the increasing overhead in good circuit simulation.

Circuit partitioning

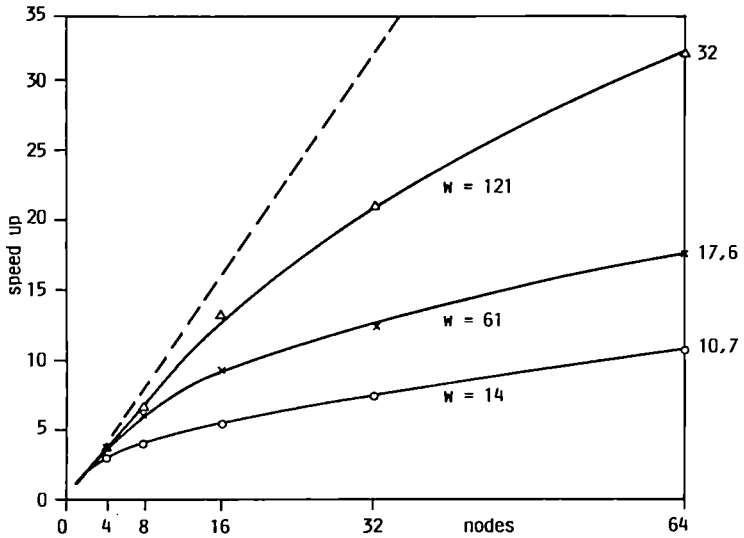
We implemented a prototype for a distributed (object-oriented) version of SILKE, in which the objects are subcircuits. Each subcircuit maintains a local clock and has its own processor. All processors run independently. Synchronization is by using a variant of Jefferson's "time-warp" algorithm [6].

This work is in an early stage. The version is used as a testbed for testing the "time warp" method in combination with circuit partitioning.

Algorithm partitioning

We have designed a 3-stage pipeline for the algorithm level compute engine. The design is based on Fairchild CLIPPER, an advanced RISC-type processor. The pipeline is implemented using the novel cache features of CLIPPER.

This engine serves as the basis for a dedicated-simulation-accelerator and for detailed pipeline design studies.



circuit	faults	fault sim. on single node	good circuit sim.
o	500	1.2 h	0.085 h
×	1212	5.2 h	0.085 h
△	3470	27.3 h	0.22 h

$$W_s = \frac{\text{fault sim. time}}{\text{good circuit sim. time}} \quad \text{(on single node)}$$

$$W_n = W_s/n \quad (n = \text{number of nodes})$$

Figure 10: Performance measurements on the parallel fault simulator

References

[1] Abramovici, M., Levendel, Y.H., Menon, P.R.
A Logic Simulation Machine; IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 2 (1983), pp. 82-94

[2] Albert, I., Mueller-Schloer, C., Schwaertzel, H.
CAD-Systeme fuer die industrielle Rechnerentwicklung; Informatik-Spektrum (1986)9, pp. 14-28

[3] Aposporidis, E., Daue, T., Mehring, P.
Structure of a multi-level simulator exploiting maximal concurrency, ESPRIT 415, Doc. No. AEG 008-86, April 1986

[4] Aposporidis, E., Jud, W.
Logik- und Fehlersimulation kundenspezifischer Schaltungen; 10. Intern. Kongress Mikroelektronik, Muenchen, 9.-11. Nov. 1982, pp. 414-423

[5] Blunden, D.F.
Boyce, A.H.
Taylor, G.
Logic Simulation - Part 1;
The Marconi Review, Vol. XL, No. 206,
Third Quarter 1977, pp. 157-171

[6] Jefferson, D.
Sowizral, H.
Fast concurrent simulation using the time warp mechanism;
SCS Multiconference, San Diego, Jan. 85,
Part: Distributed Simulation, p. 63-69

[7] Jorrand, P.
Term Rewriting as a Basis for the Design of a Functional and Parallel Programming Language;
A case study: the Language FP2;
in Fundamentals of Artificial Intelligence,
LNCS 232, 1986

[8] Leinwand, S.M.
Process oriented Logic Simulation;
IEEE, 18th Design Automation Conference,
Paper 25.1, 1981, pp. 511-517

[9] Lohnert, F.
Necessary Language Constructs for the Object-oriented Language with Respect to Logic Simulation:
ESPRIT 415, Doc. No. AEG 001-87, Febr. 1987

[10] Schaefer, P.
Schnoebelen, Ph.
Specification of a pipelined event driven simulator using FP2;
PARLE-Parallel Architectures and Languages Europe,
Vol. 1: Parallel Architectures. Eindhoven,
June 15 - 19, 1987, pp. 311 - 328

[11] Seitz, C.L.
Concurrent VLSI Architectures;
IEEE Transactions on Computers, Vol. c-33,
No. 12, (December) 1984, pp. 1247-1265

[12] Unger, B.
Lomow, G.
Andrews, K.
A process oriented distributed simulation package;
SCS Multiconference; Part: Distributed Simulation,
January 1985, San Diego, S. 76-81

[13] Odijk, E.
The DOOM system and its applications;
PARLE-Parallel Architectures and Languages Europe,
Vol. 1: Parallel Architectures. Eindhoven,
June 15 - 19, 1987, pp. 461 - 479

Project No. 415

AN OVERVIEW OF DDC: DELTA DRIVEN COMPUTER.

R. Gonzalez-Rubio, J. Rohmer, A. Bradier.
BULL SA CENTRE DE RECHERCHE
DSG/CRG/DMIA - PC 58 A 13
B.P. N° 3
68, Route de Versailles
78430 Louveciennes. France.

ABSTRACT

In this paper we present an overview of the DDC "Delta Driven Computer" and the state of this research project at the end of 1986.

DDC is a parallel inference computer, composed by a set of PCM (Processor, Communication Device, Memory) nodes interconnected. It is currently under design at BULL Research Center*.

From a conceptual point of view, DDC executes a language based on production rules, called VIM Virtual Inference Machine. This execution is made following the forward chaining strategy. Given a set of rules and a set of initial facts, the only mode of operation of the machine is the saturation (all conclusions are found).

VIM is an intermediate language; so, part of the project is the study of how to translate from a high level language to this intermediate language. The high level languages which we are thinking about are declarative ones (i.e. Logic Programming or Functional Programming).

The execution of VIM is possible by the DDEM Delta Driven Execution Model.The parallelism in the machine is achieved by distributing the facts among PCM nodes and by firing rules independently in each processor.

One goal of this project is to have a first prototype (including both hardware and software) at the end of 1987, so we try to prove that the parallelism handled by the model/machine has a valuable rate cost/performance.

Note: This work is partially supported by **ESPRIT Project 415.**

1 INTRODUCTION.

We present here the general ideas of DDC "Delta Driven Computer" and the state of this research project at the end of 1986.

DDC is a parallel inference multiprocessor computer, currently under design at BULL Research Center; early papers are just describing just part of our ideas [Gon 85], [Gon 86].

The architecture of DDC can be viewed as a multiprocessor system composed by a set of PCM (Processor, Communication device, Memory) nodes interconnected, in which there is no need to have a shared memory.

From a conceptual point of view, DDC executes a language based on production rules, called VIM (Virtual Inference Machine). This execution is made following the forward chaining strategy. Given a set of rules and a set of initial facts, the only mode of operation of the machine is the saturation (all conclusions are found).

VIM is an intermediate language; so, in the project, we study how to translate from a high level language to this intermediate language. The high level languages which we are thinking about are declarative ones (i.e. Logic Programming or Functional Programming).

The execution of VIM is possible with the DDEM (Delta Driven Execution Model). In this model the execution is driven by the facts deduced by the rules. We call these "new" facts the Delta.The parallel architecture we propose can support the DDEM.

The parallelism of the machine is achieved by distributing the facts among PCM nodes and by firing rules independently in each processor.

The implementation of DDEM is based on relational operations, so VIM rules are transformed into a program in DDCL (Delta Driven Computer Language).

One goal of this project is to have a running prototype (including both hardware and software) at the end of 1987. We had to take some decisions, and we made some restrictions to implement this prototype, but the ideas included it must prove that the parallelism handled by the model/machine has an interesting cost/performance rate.

In section 2 we give a general overview of the project DDC, the motivations for its architecture, and the key ideas concerning the different levels of languages . Section 3 presents the architecture and the DDEM mapped into it. Section 4 gives the current ideas of the first implementation. Section 5 gives some conclusions and an open for the future work.

2 A GENERAL OVERVIEW OF DDC.

2.1 MOTIVATIONS.

The basic motivations of the DDC project have been to design an efficient computer dedicated mainly to symbolic computation, to build "large" Artificial Intelligence (AI) applications.

Let us start by analyzing the situation in eighties. In one hand, the needs for high efficiency in AI are clear. It is commonly thought that they are from 100 Mlips to 1 Glips (see [Mot 84]).

But to reach such performances only with technological progress offered by Ultra Large Scale Integration is not possible. This is due to the von Neumann architecture model which is characterized by its sequential operations. Effectively, we know that the execution time of the instruction of a machine is bound by the propagation time of signals. Then predictions are that a processor's clock period cannot be smaller than 1 ns. Hence, the maximum performances are under the order of the Giga instructions per second [Lun 85]. Thus the only issue is to design parallel architectures.

On another hand, the requirement of AI machines is to maintain software costs as low as possible. This could be feasible by using a simple high level language where parallelism is hidden for the programmer.

Consequently, a prerequisite to the design of a specialized AI parallel architecture is an execution model well adapted to handle parallelism and symbolic processing.

Another prerequisite is that the architecture must not be dedicated to one particular language. Instead, it is enough opened to accommodate a variety of programming declarative styles, including:

- relational + deduction
- logic
- functional.

Typical applications for this architecture stem from relational databases, deductive databases, expert systems, simulation systems, etc..

2.2 AN INTERMEDIATE LANGUAGE APPROACH.

The language levels in DDC are:

- a high level language, which is issued of logic programming or functional programming. Currently we just work in the logic programming side.

- an intermediate language, which is the key of our approach. This language is based on production rules with a forward chaining (saturation) strategy. We call this language VIM.

- to execute the saturation the DDEM is provided. This model can be parallelized, as explained later. In fact a VIM program is translated into a program DDCL (Delta Driven Computer Language). DDC machine language is DDCL, and a program is executed following the DDEM.

The translation from a logic programming language into a VIM program is done following an algorithm called the Alexander Method, proposed by J. Rohmer [Roh 85].

In fact we define these three levels of language to get a better understanding of how the system works.

In the next sub-sections we describe the VIM language, to present how a computation takes place. Then, we examine the Alexander Method to show how a logic program is translated into a VIM program. We also present the DDEM, DDCL and the architecture of the machine.

We want to underline that what we present in the next sections are just the basic ideas of how to implement DDC, and at the end we consider how the machine can be used.

2.2.1 Virtual Inference Machine VIM.

The design of VIM is based on our background experience on production systems. [Pug 85]).

Basically, this language is composed of production rules [Pug 86], i.e. rules of the form:

$h_1 \Rightarrow c_1, \ldots\ldots c_p$
$h_1, h_2 \Rightarrow c_1, \ldots\ldots c_p$

Where the H_i and c_j are predicates of the form:

$p(x_1, \ldots\ldots, x_n)$

Where x_i is either an atom (constant) or a variable.

This means that functions (or trees) are not visible at this level.

We impose that variables in the conclusions must appear also in the hypotheses.

We can remark that the restrictions that we imposed to VIM are the same than these of Datalog (Logic database).

This means that if initially there exists a set of facts composed by constants, all the generated facts will also be composed by constants.

For implementation reasons we impose a maximum of 2 hypotheses in rules.

The only mode of operation of the machine is the saturation: Given a set of clauses and a set of initial facts "Find all possible conclusions".

The computation model is based on the notion of saturation of a rule-set by a set of

facts. This notion corresponds to the generation of the semantic model associated to the logic program made of clauses . This model is indeed the Least Fixed Point of the set of clauses (rules and facts are just clauses in First Order Logic).

Example.

Consider the following rules:

```
father(X,Y) => ancestor(X,Y)
ancestor(X,Y),ancestor(Y,Z) => ancestor(X,Z)
```

Consider the set of initial facts:

```
father(1,2)
father(2,3)
father(3,4)
```

When a saturation takes place, all the ancestors are deduced.

```
ancestor(1,2)        ancestor(1,3)
ancestor(2,3)        ancestor(2,4)
ancestor(3,4)        ancestor(1,4)
```

The saturation stops when no more facts can be deduced. The termination of the saturation process was proved by J.M. Kerisit [Ker 86] in a Datalog framework.

The saturation process in the case of commutative rules can be executed in parallel.This means that the rules can be fired at the same time, as data are available.

We can describe the DDEM in an informal way at the VIM level. When a rule is applied, then eventually a fact or a set of facts is deduced, in our terminology a BΔ (read Black Delta). Only those facts which are not contained in the database are considered as new ones or in our terminology as a WΔ (read White Delta), and this WΔ is inserted in the database, then the rules can be tried again using just the WΔ as trigger.

Example:

Consider the set of facts and rules of the previous exemple:

We consider that the initial facts are WΔ:

The following facts can be deduced when the first rule is applied:

```
ancestor(1,2)          ancestor(2,3)
ancestor(3,4)
```

As they do not already exist, they are thus inserted into the database, and then they are considered as a WΔ.

Now the following facts can be deduced from the facts in the database and the WΔ are:

```
ancestor(1,3)          ancestor(2,4)
ancestor(3,1)
```

etc.

Until any more Δ is produced.

The rules have been triggered in parallel.

In some way VIM can be considered as a sub-set of Prolog, the Datalog part of Prolog, although the execution of VIM is different with respect to Prolog, but one big advantage is that VIM is really declarative. To illustrate this point consider this example.

The following set of clauses in Prolog loops

```
father(louis,jean).
ancestor(X,Y) :- father(X,Y).
ancestor(X,Z) :- ancestor(X,Y),ancestor(Y,Z).
```

with the query:

```
ancestor(X,Y)?
```

but if it executed in saturation with VIM rules it will stop.

2.2.2 The Alexander Method.

Forward chaining as in VIM exhibits an interesting property of simplicity. But forward chaining has the drawback of computing all possible solutions to all possible predicates included in the set of rules. For instance, if we want to know the ancestors of Jean, it is useless to start by computing all the ancestors of everybody (by saturating the database) and to select afterward just the ancestors of Jean.

The Alexander Method is an algorithm to tranform a set of VIM rules (to be executed in forward chaining) and a query (to be executed in backward chaining), into a new set of rules to be executed in forward chaining, that compute just the desired solutions. In some way, the Alexander Method permits to simulate backward chaining into forward chaining.

In an informal way the Alexander Method cuts a recursive goal in:

- one problem
- one or several solutions.

For instance, the goal (as the literal) `ancestor(W,jean)` is cut in:

- a new literal: `problem_ancestor(jean)` which can be interpreted as "The problem of finding the ancestor of Jean exists"

- literals like `solution_ancestor(louis,jean)` which can be interpreted as "Louis is a solution to the problem `problem_ancestor(jean)`".

To go from backward chaining to forward chaining, we need rules which handle `problem_ancestor` and `solution_ancestor` literals.

For instance:

```
problem_ancestor(X),q => r
```

can be read as "if there is the problem of finding the ancestors of `X`, and `q` is true, then ... "

and `a => solution_ancestor(W,X)`

can be read as "if `a` is true then `W` is a solution ".

With these intuitive ideas in mind, let us process an example step by step:

Let's have as goal `ancestor (W,jean)` and the rules:

R1: `father(Y,X) => ancestor(Y,X)`
R2: `father(Z,Y),ancestor(Y,X) => ancestor(Z,X)`

R1 gives:

R1.1: `problem_ancestor(X),father(Y,X) => solution_ancestor(Y,X)`

"if there is the problem of finding the ancestors of `X`, and if `Y` is the father of `X`, then a solution is `Y`"

R2 gives:

R2.1:
```
problem_ancestor(X),father(Z,Y),ancestor(Y,X) =>
                    solution_ancestor(Z,X)
```

"if there is the problem of finding the ancestors of `X`, and if `Z` is the father of `Y`, and if `Y` is an ancestor of `X`, then a solution is `Z`"

But this rule contains itself the goal `ancestor(Y,X)`, thus it must itself be transformed. This goal will itself be cut into two pieces, yielding two new rules R2.2 and R 2.3.

R2.2: `problem_ancestor(X),father(Z,Y) => problem_ancestor(Y)`

"if there is the problem of finding the ancestor of `X`, and if `Z` is the father of `Y`, then there exists the problem of finding the ancestor of `Y`, because `Y` is an ancestor of `X`."

This rule generates a new `problem_ancestor`, which, through rule R1.1 for instance, will generate news `solution_ancestor`.

R2.3: `solution_ancestor(Y,X) => solution_ancestor(Z,X)`

"the solution to the `Y` problem are also solutions to the `X` problem".

In fact, rule R2.3 does not respect a restriction of VIM (predefined variables), since `Z` appears in conclusion and not in hypotheses. Thus, it is necessary to transmit the information `Z` between rules R2.2 and R2.3. For that purpose, we create a new predicate named `continuation`.

The final version of R2.2 and R2.3 is now:

```
R2.2':   problem_ancestor(X),father(Z,Y)=>
                    problem_ancestor(Y),continuation(Y,Z)
R2.3': solution_ancestor(Y,X),continuation(Y,Z)=>
                    solution_ancestor(Z,X)
```

The detailed algorithm of the Alexander Method was presented in [Roh 86].

2.2.3 Delta Driven Execution Model DDEM.

The DDEM is an algorithm to execute the saturation upon a logic database.

Considering a rule R_i containing the predicate `r` in the conclusion. Each time that a set of facts are deduced from the execution of the rule R_i, we call them a BΔ. This BΔ is compared with facts that already exist. If a fact exists, nothing happens. If it does not, then it is considered as a WΔ. Then this WΔ is inserted into the data base, and it is used to eventually fire rules with predicate `r` in hypotheses.

A VIM rule as

```
p,q => r
```

is transformed into:

```
WΔp,qc => BΔr1
WΔq,pc => BΔr2
```

where q_c and p_c are the current representation of `q` and `p` in the database the $B\Delta r$ is:

```
BΔr <- BΔr1 Union BΔr2
```

where <- is the assignation and `Union` is a set operation.

To eliminate the duplicates we consider that r_c means the current representation of `r` in the database

$W\Delta r$ <- $B\Delta r$ - r_c

where "-" is a set operation.

The insertion of $W\Delta r$ into r_c is represented by:

r_c <- r_c Union $W\Delta r$

this $W\Delta r$ can be tried in rules of the form:

$W\Delta r, s$ => t

when any more $W\Delta$ is produced the logic database is saturated, and the work is finished.
Let us apply this algorithm to an exemple.

Consider the following rules:

father(X,Y) => ancestor(X,Y)
father(X,Y),ancestor(Y,Z) => ancestor(X,Z)

Consider the initial facts:

father(1,2) father(2,3) father(3,4)

So the rules are transformed into:

R1: $W\Delta$father(X,Y) => $B\Delta$ancestor.$_0$(X,Y)
R2: $W\Delta$father(X,Y),ancestor$_c$(Y,Z) => $B\Delta$ancestor.$_1$(X,Z)
R3: father$_c$(X,Y),$W\Delta$ancestor(Y,Z) => $B\Delta$ancestor.$_2$(X,Z)

then the saturation is executed as follows:

Init:
  ancestor$_c$ <- empty

all initial facts are put in father$_c$

  father$_c$ <- father(1,2),father(2,3),father(3,4)

also all initial facts are considered as $W\Delta$

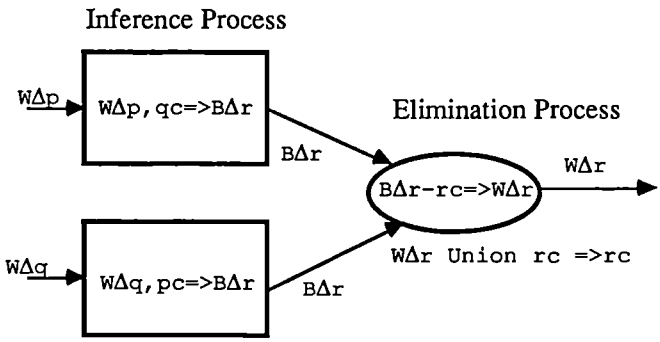
  $W\Delta$father <- father(1,2),father(2,3),father(3,4)
Label 1:
  Apply $W\Delta$ to rules R1, R2, R3
  $B\Delta$ancestor <- $B\Delta$ancestor.$_0$
                      Union $B\Delta$ancestor.$_1$
                      Union $B\Delta$ancestor.$_2$
  $W\Delta$ancestor <- $B\Delta$ancestor - ancestor$_c$
  ancestor$_c$ <- ancestor$_c$ Union $W\Delta$ancestor
  if ($W\Delta$ancestor is not empty) goto Label 1

Another way to represent what happens when saturation takes place is with an execution graph.

We can differentiate two types of process:

- application of rules.
- elimination of duplicates.

We can represent as a square block the application of a rule, and as a round block the elimination. See figure 2.1. In the figure a square block produces a BΔr, and a round block produces a WΔr. As many rules can produce conclusions on predicate r, all the outputs BΔr go to the round block where elimination of r facts takes place.



The DDEM process.

<<<<Figure 2.1>>>>

The elimination process "transforms" a BΔ into a WΔ if the BΔ is not in the database.

Figure 2.2 shows the execution of a saturation in the form of a graph. Arrows represent the Δ productions. We can see that execution is driven for the Δ. We know that as far as WΔ are produced, the execution goes on but when there is no more WΔ produced, saturation stops.

To represent what is happening in the elimination process, we define two operations; at this level they can be represented as rules. The first is the elimination:

```
BΔr - rc => WΔr
```

The second is the insertion of the WΔ into the database:

```
WΔr Union rc => rc
```

BΔr is the BΔ on r, r_c contains the facts on predicate r, WΔr is the WΔ on r.

This two rules are not VIM rules, they are needed for the DDEM.Their semantic is then different from VIM rules.

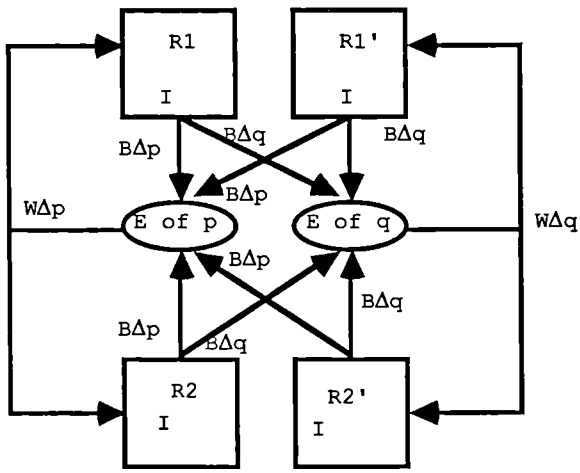
We can remark that a BΔ arrives at any time the operation `BΔ <- BΔ.1 Union BΔ.2`

becomes implicit and it is unnecessary to add the suffix to distinguish BΔs.

One advantage of this model is that it is asynchronous, in the case of monotonic and commutative rules. This means that the order of Δ arrivals does not modify the final result.

Rules

```
R1: WΔp(X,Y),q(Y,Z) => BΔp(X,Z), BΔq(X,X)
R1': p(X,Y),WΔq(Y,Z) => BΔp(X,Z), BΔq(X,X)
R2: WΔq(X,Y), p(Y,Z) => BΔq(X,Z), BΔp(Z,Z)
R2': q(X,Y),WΔp(Y,Z) => BΔq(X,Z), BΔp(Z,Z)
```



DDEM Graphe.

<<<<Figure 2.2>>>>

2.2.4 Delta Driven Computer Language DDCL

To fill the gap between VIM code and machine language of each node we define another language called DDCL. The primitives of the language are mainly relational operations.

Some examples of rules:

```
p(X,'jean') => q(X)
```

can be implemented as a selection in the relation p.

```
p(X,Y),q(Y,Z) => r(X,Z)
```

can be implemented as a join beween relations p and q.

In section 3 we specify the reasons of this choice, and we present the execution of the primitives.

This transformation of rules VIM into relational operation is consistent with the defined properties of DDEM.

So we can consider that at DDCL level instead of predicates and facts the machine handles relations and tuples.

3 THE ARCHITECTURE AND DDEM.

3.1 THE ARCHITECTURE OF DDC.

DDC consists in a set of nodes linked by an interconnection system without shared memory.

All nodes are identical, a P-C-M triple: Processor, Communication device and Memory.

The Processor has two parts:

- a general purpose microprocessor: Motorola 68020
- a special purpose custom VLSI chip called µSyC.

µSyC chip acts as a coprocessor of the 68020. This means that when a coprocessor code is detected by the 68020, it "calls" the coprocessor to execute a coprocessor instruction. Each instruction of the coprocessor is a complete relational operation, the whole algorithm is microprogrammed.

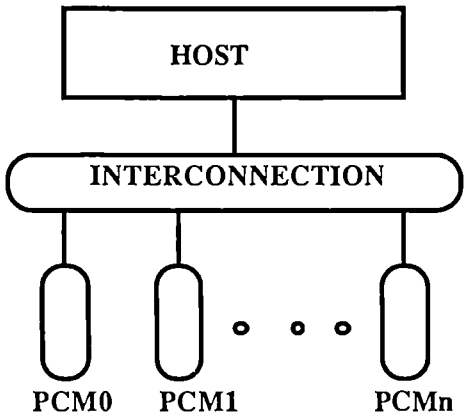
The Memory can be divided into three parts :

- fast static RAM on the CPU board - but not cache
- boards with large capacity
- a secondary storage.

The Communication Module.

This module is responsible for receiving and sending messages from and to the interconnection network.

Figure 3.1 shows the DDC architecture.



DDC Architecture.

<<<<Figure 3.1>>>>

3.2 DDEM INTO DDC.

The mapping strategy of DDEM into the DDC architecture is statically determined. We try to balance the load in the machine, keeping the communications as low as possible.

Here we detail how DDEM is mapped into DDC architecture without care about the initialization phase. So at a given time of a saturation we can consider that:

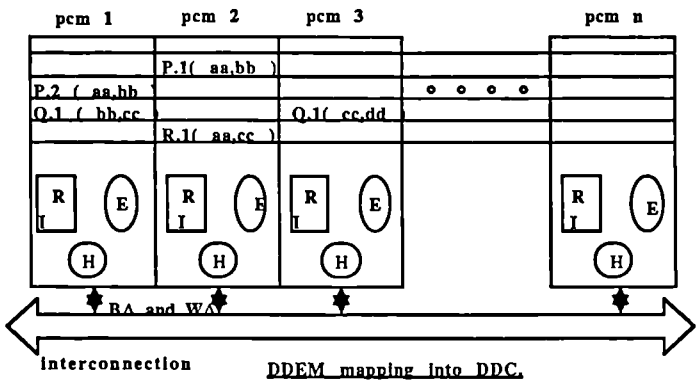
- facts are distributed along the pcm nodes
- all rules of DDEM level are copied on each node.

To be more precise:

- relations are distributed along the pcm nodes. According to a hash function h determined at compile time
- the DDCL code which "makes" the application of rules is copied on each node
- the code of the hash function, which serve to distribute relations is copied on each node.

As show on figure 3.2.

In P.1 and P.2 are the same facts, but P.1 and P.2 are distributed in different ways.



<<<<Figure 3.2>>>>

A relation is distributed according to an hash code function applied to the value of one of their attributes. Relations can be "duplicated" in case of uses upon different attributes in rules.

In the case of "duplicate" relations they are distinguished by adding a suffix to the name of the relation.

The advantage of this mapping is that: the data go where they will be used, the locality is ensured by the hash function statically defined.

This is a commented example of a saturation following the DDEM:

Consider that DDC is composed of three nodes.

And the set of two VIM rules is :

R1: `p(X,Y),q(Y,Z) => r(X,Z)`
R2: `p(X,Y),r(X,Z) => p(Y,Z)`

The set of four initial facts:

```
p(aa,bb), q(cc,dd), q(bb,cc), r(aa,cc)
```

that must be stored as relations as follows:

At compilation it could be noticed that the `p` relation is used in R1 and R2 but in each one according to different attributes.

So, in the machine, instead of having `p` , there are two copies: `pc.1` and `pc.2`.

`pc.1(X,Y)` to be used by R2 distributed according to values of its first argument.

`pc.2(X,Y)` to be used by R1 distributed according to values of its second argument.

For q there is just $q_{c.1}$ distributed according to values of its first argument.

For r there is just $r_{c.1}$.

So if we apply the hash function to values of arguments of the initial facts, we identify in which processor a tuple will be stored.

```
H(1,p.1(aa,bb)) = H(1,r.1(aa,cc)) = h(aa) = 1
H(2,p.2(aa,bb)) = H(1,q.1(bb,cc)) = h(bb) = 2
H(1,q.1(cc,dd)) = h(cc) = 3
```

where H is a function, with the arguments: the number of the attribute to apply the function and a tuple; h is the hash function which has as argument the value of the selected argument, and returns the identity of the destination node, and 1,2,3 are the node PCM number.

So in node PCM 1 there exists:

$p_{c.1}$(aa,bb) $r_{c.1}$(aa,cc)

In node PCM 2 there exists:

$p_{c.2}$(aa,bb) $q_{c.1}$(bb,cc)

In node PCM3 there exists:

$q_{c.1}$(cc,dd)

We can consider that a saturation starts here:

The initial facts are WΔ:

Arriving to PCM 1:

WΔp.1(aa,bb) WΔr.1(aa,cc)

Arriving to PCM 2:

WΔp.2(aa,bb) WΔq.1(bb,cc)

Arriving to PCM 3:

WΔq.1(cc,dd)

The compilation of rules are:

R1 gives

R11 WΔp.2(X,Y), $q_{c.1}$(Y,Z) => BΔr.1(X,Z)
R12 WΔq.1(Y,Z), $p_{c.2}$(X,Y) => BΔr.1(X,Z)

R2 gives

R21 WΔp.1(X,Y), $r_{c.1}$(X,Z) => BΔp.1(Y,Z)
R22 WΔr.1(X,Z), $p_{c.1}$(X,Y) => BΔp.1(Y,Z)

In this example the elimination is done with respect to one copy of each relation, here

we chose to execute the elimination taking as a reference the relation indice 1.

Elimination rules are:

R3: BΔp.1(X,Y) - pc.1(S,T) => WΔp.1(X,Y),WΔp.2(X,Y)
R4: BΔr.1(X,Y) - rc.1(S,T) => WΔr.1(X,Y)

We recall that in R3 and R4 the condition to produce something is X≠S or Y≠T.

R5: WΔp.1 Union pc.1=> pc.1
R6: WΔr.1 Union rc.1=> rc.1

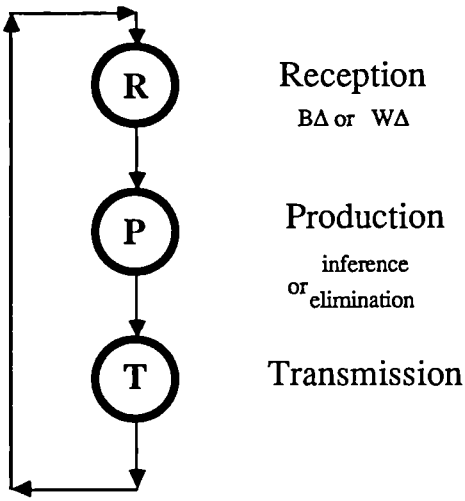
To each produced tuple of a BΔ one node the hash fonction h is applied, as shown before. Then each tuple is sent to just one processor.

In the elimination rules we can notice that in some cases one WΔ appears as conclusion (R4) and in other cases more than one WΔ appear. What happens is that each produced tuple, must be sent to one (i.e. R3) or more processors (i.e.R4) applying the hash function to different attributes.

Note that the processor who receives a tuple (a BΔ) contains the part of the relation where the tuple (a BΔ) can be "transformed" into a WΔ.

Inside each node, the execution mechanism can be divided into basic cycles triggered according to delta arrivals. See figure 3.3.

R reception of WΔ or BΔ into a buffer
P production which is either an inference
WΔ produces a BΔ or nothing or an elimination
BΔ produces a WΔ or nothing
T transmission of either WΔ resulting from elimination or BΔ resulting from inference.



Execution mechanism in a node.

<<<<Figure 3.3>>>>

Let us see in details what happens in one node when a saturation starts.

At T0: data are distributed and "rules" installed.

At T0 + the time to recognize the WΔ:

In node PCM1.

Node state : facts = { $p_{c.1(aa,bb)}$, $r_{c.1(aa,cc)}$ }

Production by WΔp.1(aa,bb) and R21
of a BΔ containing {p (bb,cc)}.
Transmission of this BΔ to node PCM 2
because H(1,BΔp.1(bb,cc)) = h(bb) = 2
Production by WΔr.1(aa,cc) and R22
of a BΔ containing {p (bb,cc)}.
Transmission of this BΔ to node PCM 2
because H(1,BΔp.1(bb,cc)) = h(bb) = 2

Then this node waits for a WΔ or a BΔ or for the end of the saturation.

The work on the other nodes follows the same sequence, as far as there are Δ in the input of a node.

We call this implementation of DDEM: scenario with elimination with respect to a

single file, because the elimination is made taking as a reference one representation of the predicate. Another scenario where the elimination is made taking each "copy" as a reference is also under study.

3.3 DDCL AND DATA REPRESENTATION.

As precised before, DDCL consists mainly in relational operations, most of them are executed following the principles of filtering by automata presented in [Gon 84], and the join algorithm LA-JOIN presented in [Bra 86].

Let us first describe how a selection can take place. Suppose that we make the selection on relation `r` of the tuples where the first attribut is equal to jean or paul or pierre. We build an automaton which recognizes jean or paul or pierre. Then each tuple of `r` is sent to the automaton which indicates if the tuple has to be kept or not.

The automaton representation can be more compact than the classical matrix.

The join of two files can be executed by making a selection of tuples of the first file then building an automaton to make a selection upon the second file.

3.3.1 The relational operations in DDC.

Here we consider just a join.

If we have the VIM rule:

R1: `p(X,Y) , q(Y,Z) => r (X,Z)`

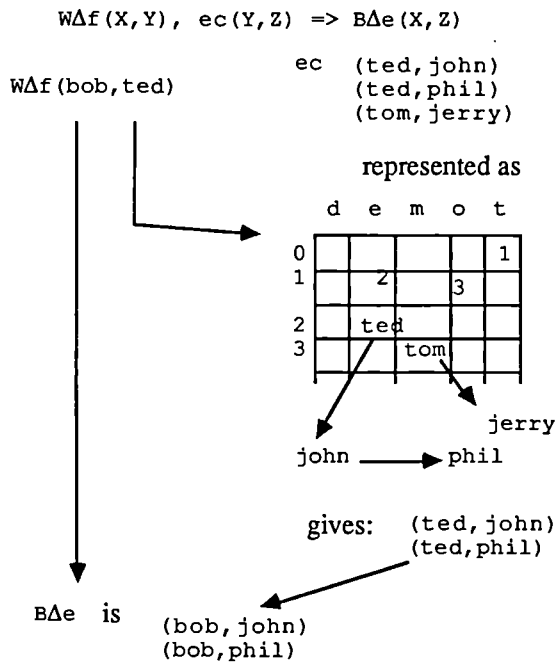
First it is transform in rules:
R11: `WΔp.2(X,Y) , qc.1(Y,Z) => BΔr.1(X,Z)`
R12: `WΔq.1(Y,Z) , pc.2(X,Y) => BΔr.1(X,Z)`

The current representation of `qc.1` will be stored as an automaton and pointers to the elements of the second attribute. The automaton is to recognize if a tuple of the `WΔp.2` can be joined to `qc.1`. If that is the case a `BΔ` is produced. This `BΔ` contains tuples with the value of the first attribute and the different values of the second attribute of `qc.1`. See figure 3.4.

The advantage of this solution is that with just a few comparaisons of characters of the `WΔ` against the automaton, a `BΔ` can be produced.

The same technique can be used to make other operations in DDC as pattern matching.

This solution is consistent with the DDEM because it is asynchronous. The locality of data is ensured by the hash code function. The `WΔ` only arrives to the node where the ith part of the join can take place.



A join in DDC.

<<<<Figure 3.4>>>>

4 FIRST IMPLEMENTATION.

With all the ideas presented in section 2 and 3, we are implementing a first version of DDC.

Even if conceptually the DDEM seems to be a good solution to parallelism, compromises for the implementation must be taken and evaluated. So the aim of this first version is to gain experience, in parallel machines and particulary in the implementation of the DDEM. We hope to obtain a feedback to be able to propose improvements.

For this first version, we take an existing multiprocessor machine a BULL SPS7, and we concentrate our efforts in the software implementation of DDCL and the DDEM on this machine.

The BULL SPS7 is a multiprocessor which uses up to 8 PM (processor and local memory); they are attached to a bus, where a common memory lets the PM communicate. The modules PM play the role of nodes PCM.

The advantage of this solution is that the BULL SPS7 has already a lot of software.

4.1 FROM A USER PROGRAM TO EXECUTABLE CODE.

One of the processor of the BULL SPS7 will act as a host, the others constitute DDC. In this configuration DDC acts as a coprocessor to execute saturations. A program consists in sequential parts grouped as blocks and parallel parts. Each block for DDC describes a saturation, it contains the set of facts and the set of rules to be saturated. The user must indicate which blocks of the program are parallel, then executed by DDC.

The order of block execution is imposed by the user program and controlled by the host. Serial blocks are executed by the host and calls are made to DDC each time a saturation is needed.

Before execution, a program is compiled by the host, in one hand the sequential blocks, and on the other hand the parallel blocks are compiled first into VIM then into DDCL.

The transformation of VIM to DDCL also gives the primitives to load DDC, to initialize a saturation and to get results.

4.2 CURRENT WORK.

Currently we are working on:

- implementation of the first version
- optimisations of the compiler from high level language to VIM
- optimisations of the compiler from VIM to DDCL
- memory management problems
- planning the integration and test of the first version
- looking for applications
- performance prediction.

5 CONCLUSIONS.

We present in this paper the basis for the design of DDC, an AI parallel machine supporting DDEM, a suitable execution model where rules are executed in parallel with relational operators.

We describe all the levels, from the high level language until machine level, and how we can go from one level to another.

We are working on this first implementation as described in section 4.

Also we are active in the possible extensions of the system , as for instance:

- DDC as a coprocessor of a main -frame.
- introduction of predefined predicates.
- list and function processing.

Finally, we would like to thank the anonymous reviewers of the paper for their comments, also to: R. Lescoeur, J. M. Kerisit, and J. M. Pugin, F. Anceau, K. R. Apt and B. Bergsten; they are involved in the project and we benefit from their comments, ideas and suggestions. We are grateful to E. Pavillon for her help, improving the English of the paper, but if there are still some English faults we assume the responsability.

6 REFERENCES.

[Bra 86] Bradier A.: "LA-JOIN: Un Algorithme de Jonction en Mémoire et sa Mise en Oeuvre sur le Filtre SCHUSS. II èmes Journées Bases de Données Avancées. Giens, Avril. 1986.

[Gon 84] Gonzalez-Rubio R., Rohmer J., Terral D.: "The SCHUSS Filter: A Processor for Non-Numerical Data Processing". 11th Annual International Symposium on Computer Architecture. Ann Arbor. 1984.

[Gon 85] Gonzalez-Rubio R., Rohmer J.: "From Data Bases to Artificial Intelligence : A Hardward Point of View". Nato Summer School, Les Arcs 1985.

[Gon 86] Gonzalez-Rubio R., Bradier A., Rohmer J.: "DDC Delta Driven Computer. A Parallel Machine for Symbolic Processing". ESPRIT Summer School on Future Parallel Computers. University of Pisa. June.1986.

[Ker 86] Kerisit J. M.: "Preuve de la Méthode d'Alexandre par une approche algébrique", BULL Rapport Interne, Mai1986.

[Lun 85] Lunstrom S. F., Larsen R. L.: "Computer an Information Technology in the Year 2000- A projection". Computer, September 1985.

[Mot 84] Moto-oka T., Stone H. S.: "Fifth Generation Computer Systems: A Japanese Project". Computer, March 1984.

[Pug 85] Pugin J.M.: "BOUM: An Instantiation of the (PS)2 concept". 5èmes Journées Internationales Systèmes Experts. Avignon 1985.

[Pug 86] Pugin J.M. : "VIM Language". Bull Internal Report 1986.

[Roh 85] Rohmer J., Lescoeur R. : "The Alexander Method. A technique for the processing of recursive axioms in deductive databases". Bull Internal Report 1985.

[Roh 86] Rohmer J., Lescoeur R., J. M. Kerisit: "The Alexander Method. A technique for the processing of recursive axioms in deductive databases". New Generation Computing, 4. 1986.

Project No. 415

A two-level approach to logic plus functional programming integration

M. Bellia+, P.G. Bosco*, E. Giovannetti*, G. Levi+, C. Moiso*, C. Palamidessi+

* CSELT - via Reiss Romoli 274 - 10148 Torino - Italy
+ University of Pisa - Dipartimento di Informatica - corso Italia 40 - 56100 Pisa - Italy

1. Introduction: the reasons for the integration

Logic programming and functional programming are the two most popular styles of declarative programming, and some debate went on in the past on the respective pros and cons of each of them with respect to the other. No wonder that an attitude to overcome this discussion by combining the two paradigms and thus the advantages of both (without their drawbacks) developed relatively soon.

Logic programming is characterized by two essential features: non-determinism, i.e. search-based computation, and logical variable, i.e. unification. The bidirectionality of unification, in contrast with the unidirectionality of pattern-matching, allows procedure invertibility and partially determined data-structures, and thus a more compact and flexible programming style. The central role of search (or *don't know* nondeterminism) in logic programming languages is connected with their being halfway between theorem provers and standard programming languages, which makes them particularly adequate for artificial intelligence applications, or as executable specification languages.

Functional languages share with logic languages the formal simplicity (function application as the only control construct) and the property of being based on a well-established mathematical theory, in this case (some kind of) lambda-calculus, which directly provides a clear semantics. Reduction is the key concept, which corresponds to the use of equalities as rewrite rules, in contrast with standard logic programming, where the equality is not explicitly present.

Apart from notational aspects, the other fundamental difference with the languages based on first-order logic is the presence of the notion of higher-order function, a powerful structuring concept which can be exploited in programming in the large, in program synthesis from specifications, etc.

In addition, functional languages offer a variety of other useful programming concepts, such as typing disciplines, outermost/innermost strategies, etc., which would be profitably included in an integrated system.

Several different approaches to the integration have been proposed. Roughly, they can be

partitioned into two classes. On the one hand, the logic+functional languages, i.e. the logic languages enhanced with some limited functional features, essentially based on first-order logic with equality: they usually lack one of the very aspects which characterize functional programming, i.e. higher-order. On the other hand, the functional+logic approach, i.e. the functional languages augmented with logic capabilities, for example the ability to solve equations, to invert functions, etc. For a survey of the most relevant proposals of both kinds see [3, 13].

The approach chosen in the Subproject D of ESPRIT Project 415, described in this paper, consists in splitting the problem into two levels. The upper level, or user interface, is a sort of superlanguage combining higher-order functional programming (i.e. lambda-calculus with a simple theory of types) with Horn clause logic. It is represented, for the time being, by the experimental language IDEAL [4]. This level is still subject to evolution, as it should eventually contain all the desirable features for this kind of language, w.r.t. different fields of applications. Some theoretical aspects still have to be deepened, in particular concerning a satisfactory definition and semantic characterization of the programming problems we want to be able to solve.

The lower level is a Prolog-like language augmented with directed equality, which is represented at the present stage of research by K-LEAF [14, 7], an evolution and refinement of the language LEAF [2]. It is a rather well assessed language, with a clear and original declarative semantics, and with a sound and complete operational semantics.

The upper level is mapped into this intermediate language by means of a transformation which removes the higher order by compiling it into first order (through a meta-interpretation). Moreover, in any implementation there will be a bottom level consisting of a control-flow imperative language close to the physical architecture (e.g. C-language in the present sequential prototype, maybe Occam in a future parallel architecture). Between this "machine-language" and the intermediate logic language further virtual machines could be introduced, corresponding e.g. to the elimination of nondeterminism in a mapping to a lower-level logic language, such as, for instance, Concurrent Prolog.

2. The Higher-Order Language

The user level is represented by the prototype higher-order (HO in the following) language IDEAL, for which an implementation on top of a standard Prolog environment is already available with a good efficiency.

The syntax for the functional component is, as usual, a sugared version of a *polymorphic lambda-calculus* with some form of universal types, in the style of MIRANDA [28] or ML, while the logical component is basically the underlying first-order (FO in the following) (Prolog-like) language.

The integration of the logic into the functional part is obtained, as at the intermediate FO level, by allowing function definitions to be heads of Horn clauses, functional terms to be arguments of predicates, and goals to be equations (to be solved). A program is therefore a set of possibly conditional function definitions, i.e. clauses of the form:

fname = λ *X1*.λ*X2*...λ*Xn.term* :- *cond.*

or equivalently

fname@X1@...@Xn = term :- cond.

where @ stands for the application. *Fname* is the name of the curried function being defined, and *Term* is a term, not starting with λ, of a lambda-calculus equipped (sugared) with the *if-then-else* construct and a number of data constructors, tests and selectors, along with the related reduction rules. Recursive definitions are allowed with their usual meanings. The condition *cond* is a conjunction of literals, i.e., in general, a conjunction of equations between lambda-terms. Definitions by cases are also possible, with a straightforward extension of the pure lambda-notation:

fname@Dt1@...@Dtn = term :- cond.

where the *Dti* are data terms.

The usual *let*, *letrec*, *where*, *whererec* structuring constructs are permitted with their standard meanings corresponding to pure lambda-calculus expressions. The following example of functional program taken from [28] is a good introduction to the syntax (the type information inferred by the system is listed after the ":" symbol).

Example 1.

```
foldr@Op@Z = g
  where    g@[] = Z,
           g@[A|X] = Op@A@(g@X).          :(A-->A-->A)-->A-->(list(A)-->A)
product = foldr@(*)@1.                    :list(int)-->int
sum = foldr@(+)@0.                        :list(int)-->int.
sum@[1,2,3]                ;term to be evaluated
:int          6            ;result
```

The user is allowed to define its own polymorphic data types, through a construct like the one suggested in [28]:

*typename(Typevar1,...,Typevarn) ::= constructor1(Typename11,...,Typename1*n_1*);*

.....

constructorm(Typenamem1,...,Typenamemn$_m$*)*

which declares data objects of type *typename*, the structures being identified by *constructor1*, ...,*constructorm*, each of the appropriate argument types.

If the possibility of currying is to be coherently extended to predicates, these have to be considered as boolean-valued functions. The standard definition:

p :- a1,...,an

has to be regarded as a short notation for:

p = true :- a1 = true,...,an = true.

With this extension of higher-order capabilities to predicates, definitions of predicate combinators and lexically scoped predicate definitions become possible, which greatly improves conciseness and modularization of logic programs, as is suggested by the following example.

Example 2.

```
comb@P@X@Y :- P@Z@X,P@Z@Y.          :(A-->B-->bool)-->A-->B-->bool
non-disjoint = comb@member
   where   member@X@[X|L] :-.
           member@X@[Y|L] :- member@X@L.   :list(A)-->list(A)-->bool

person ::= (a;b;c;d;...)                  ;declares type person
brothers = comb@parent
   where   parent@a@b :-.
           parent@c@d :-.
                .....                              :person-->person-->bool
```

where *non-disjoint* is a predicate which succeeds when two lists are non disjoint, while *brothers* succeeds when its two arguments have a common parent.

The main program, or goal, is either the application of a function to arguments, i.e.

?- f@t1@...@tn [= Result]

or an existential logic-programming goal:

?- p@t1@...tm [= true]

or, more generally, an equation between lambda-terms

?- t1 =t2.

As for the integration primitives, the construct *Term such_that Cond* has been introduced as a functional-flavoured version of the logic-programming conditional definition *...Term ...:- Cond...* For example,

```
k = g(X such_that p(X)).
```

is equivalent to:

```
k = g(X) :- p(X).
```

Moreover, a metalevel construct *bag* is available for producing, in formally one step, the possibly empty list of all the *Term* satisfying *Cond*. The following definition of *quicksort*, derived by a widely known functional program built on functional set-expressions, is an example of the use of *bag*, which enables a more abstract specification of the algorithm with respect to the way the list is scanned.

Example 3.

```
[] ++ L = L                ; definition of append
[X|U] ++ L = [X|U++L].                    :list(A)-->list(A)-->list(A)

gt@X@Y :- X > Y.                          :A-->A-->bool
lt@X@Y :- X < Y.                          :A-->A-->bool
gte@X@Y :- X >= Y.                        :A-->A-->bool
lte@X@Y :- X =< Y.                        :A-->A-->bool
inv(lt) = gte,
inv(gt) = lte.                            :(A-->A-->bool)-->(B-->B-->bool)

qsort@p@[] = [],
qsort@p@[A|L] = qsort@p@bag(X,(member(X,L),p@X@A)) ++
                    [A|qsort@p@bag(X,(member(X,L),inv(p)@X@A))]
                                          :(A-->A-->bool)-->list(A)-->list(A)
```

An example of the symmetrical possibility, i.e. the logic part "calling" the functional one, is the following goal, where the predicate *member* "calls" the function *map*, whose application occurs in one of the predicate arguments:

Example 4.

```
map([],F) = [],
map([X|L],F) = [F@X|map(L,F)]                    :list(A)-->(A-->B)-->list(B)

X such_that member(X,map([1,2,3],lambda(x,x+1)))   ;term to be evaluated

:int   2;
       3;
       4
```

the values 2,3,4 are "alternative" values which in a sequential environment are obtained by "backtracking".

Finally, to get the flavour of a more exciting field of application, consider the simple goal

F@[1,1] = [1,1]

executed in the scope of the definitions of functions like reverse, append, the previous qsort. Among the possible solutions we can find

F = lambda x.x; F = lambda x.[1,1]; F = reverse; F = append@[]; F = qsort@lt

while the goal: F@[1,2] = [2,1]
would only yield the solutions

F = lambda x.[2,1]; F = reverse; F = qsort@gt

HO programs, where functions and predicates are higher-order in the sense that they can take as arguments and deliver as results functions and predicates, are converted into "equivalent" L+F programs which are first-order in the usual logic sense (no quantification over predicates is allowed), e.g. K-LEAF programs (see next section). This merely amounts to considering the HO level as an object language described in the FO language used as a metalanguage. Terms of the object language, i.e. lambda-terms, are constants of the metalanguage. Reduction rules become FO equational axioms for the object structures in the metalanguage. "Equivalence" between the original program and its FO translation means that the transformation has to be sound and complete w.r.t. the class of solutions of equations we are interested in at the HO level. This strongly depends on the relation between the execution strategy of the lower level and the extension of the runtime core which is put at this same level to support the upper language. Some mappings from IDEAL to K-LEAF are complete (and sound) for a limited but meaningful class of programming (unification) problems, while for broader classes a complete mapping would involve a too large amount of interpretation with the related inefficiency. Enhancements of K-LEAF are presently being studied in order to be able to cope with this kind of problems efficiently.

3. The First-Order language

The lower level is a FO logic+equational programming language, i.e. a language based on HCL with equality. The term syntax is any *signature with constructors*, i.e. it is based on the distinction between constructors and functions, corresponding to the distinction, found in all the ordinary programming languages, between data structures and algorithms. Among the other distinctive features there are the allowing for partial functions and infinite objects, joined to a complete semantic characterization.

The concrete syntax of the first-order L+F programming system is basically the same as the language LEAF [2]. The alphabet consists of a set $\boldsymbol{C}$ of data constructor symbols, a set $\boldsymbol{F}$ of function symbols, a set $\boldsymbol{P}$ of predicate symbols, a set $\boldsymbol{V}$ of variable symbols, and the special equality symbol =. The distinction between data constructor and function symbols leads to the distinction between *(general) terms* and *data terms*. The former are built both on $\boldsymbol{C}$ and $\boldsymbol{F}$ (and $\boldsymbol{V}$), the latter are built on $\boldsymbol{C}$ (and $\boldsymbol{V}$) only. The clauses of the language are defined in the usual way, with some constraints on the syntax of atoms.

A *head atom* is:

i) a *functional head* $f(d_1,...,d_n) = t$, where f is a function symbol, $d_1,...,d_n$ are data terms, t is a term, and the following two conditions are satisfied:

1) *left-linearity*: multiple occurrences of the same variable in $(d_1,...,d_n)$ are not allowed

2) *definite outputs*: all the variables occurring in t must also occur in $(d_1,...,d_n)$

ii) a *relational head* $p(d_1,...,d_n)$, where p is a predicate symbol and $d_1,...,d_n$ are data terms.

A *body atom* is:

i) an *equation* $t_1 = t_2$, where t_1 and t_2 are terms

ii) a *relational atom* $p(t_1,...,t_n)$, where p is a predicate symbol and $t_1,...,t_n$ are terms.

A *program* W is a set of definite clauses $\{C_1,..., C_m\}$ such that:

for each pair of equational heads $f(d'_1,...,d'_n) = t'$ and $f(d''_1,...,d''_n) = t''$,

$f(d'_1,...,d'_n)$ and $f(d''_1,...,d''_n)$ are not unifiable (*superposition free*).

The above syntax, unlike the earlier version of LEAF, is to be considered a sugared form of the actual underlying language which we called Kernel-LEAF, or K-LEAF, where all the equalities in the bodies and in the goal are *strict equalities*, denoted by the symbol $\equiv$, while equalities in the heads always are non-strict equalities, denoted by the usual symbol =.

In K-LEAF the heads of the clauses, either functional or relational, must be *left-linear*, and a user-language clause like $p(x,x) \leftarrow B_1,...,B_n$ is automatically transformed by the parser into the K-LEAF clause $p(x,y) \leftarrow x \equiv y, B_1,...,B_n$. This transformation cannot be used for functional heads, since it would cause the loss of the superposition-freedom. In the functional case the

left-linearity constraint has been therefore introduced directly in the user-language

It is worth noting that equalities in the body, multiple occurrences of the same variable in (the argument-part of) a head or functional nestings in (the argument-part of) a head would introduce a not-semidecidable equality test on (possibly non terminating) subterms. Moreover, from the point of view of the declarative semantics developed in next subsection, this would violate the continuity requirement for the interpretations of functions and predicates.

The other constraints, i.e. definite outputs and superposition-freedom, have to do with confluence, i.e. with the requirement that function definitions actually define (univocal) functions.

The following set of clauses is a correct K-LEAF program:

Example 5.

```
samefringe(t1,t2) :-  fringe(t1) ≡ fringe(t2).
fringe(t) = fringe1(t,nil).
fringe1(leaf(x),continuation) = cons(x,continuation).
fringe1(tree(tl,tr),continuation) = fringe1(tl,fringe1(tr,continuation)).
```

3.1 . The model-theoretic semantics.

The natural setting for assigning meanings to possibly partial functions is found in the concept of algebraic CPO.

Let us briefly summarize the related notions. Let X be a set and let $\leq$ be an *order relation* on X. A set $D \subseteq X$ is a *chain* iff D is *totally ordered* with respect to $\leq$. $(X, \leq)$ is a Complete Partial Order (CPO) iff there exists a minimal element $\perp_X$ (bottom) and every chain D has a *least upper bound* $\bigsqcup D$. An element a is a *finite* (or *algebraic*) element iff there are not any infinite chains (from $\perp_X$) to a. $(X, \leq)$ is an *algebraic CPO* iff every element a is the least upper bound of the set of the *finite lower elements* (its finite approximations). Let $(X, \leq)$ and $(Y, \leq)$ be CPO's, and let f be a function from X to Y. f is *continuous* iff $f(\bigsqcup D) = \bigsqcup_{d \in D} f(d)$ holds for every chain D.

A simple algebraic CPO is the set $LL = \{ \perp_{LL}, true, false \}$, with the ordering $\perp_{LL} \leq true$ and $\perp_{LL} \leq false$. (three-valued boolean CPO). Note that all the elements of LL are algebraic. The formulae of the language will receive - from interpretations - truth-values in this CPO: we have therefore a three-valued logic, with "undefined" besides *true* and *false*.

Definition. Given a K-LEAF program W, an *interpretation* for W consists of an algebraic CPO $(X, \leq)$, and a meaning function $[\![\,]\!]_X$ which assigns to every constructor or function symbol a continuous function on X, and to every predicate symbol a continuous function from X to LL. The meaning of terms and atoms involves, as usual, the notion of *environment*. (that is a function

$\rho: V \to X$) and is derived by imposing the structural compositionality, i.e.:

$$[\![v]\!]_{X,\rho} = v\rho \qquad \text{for } v \in V.$$

$$[\![f(t_1,...,t_n)]\!]_{X,\rho} = [\![f]\!]_X ([\![t_1]\!]_{X,\rho}, ..., [\![t_n]\!]_{X,\rho}) \text{ for } f \in C \cup F,\ t_1,...,t_n \in T(V)$$

$$[\![p(t_1,...,t_n)]\!]_{X,\rho} = [\![p]\!]_X ([\![t_1]\!]_{X,\rho}, ..., [\![t_n]\!]_{X,\rho}) \text{ for } p \in P,\ t_1,...,t_n \in T(V)$$

The equality, i.e. the symbol = is interpreted as the identity eq_X on the CPO $(X, \leq)$, i.e.:

$$[\![t_1 = t_2]\!]_{X,\rho} = [\![t_1]\!]_{X,\rho}\ eq_X\ [\![t_2]\!]_{X,\rho} = \begin{cases} true & \text{if } [\![t_1]\!]_{X,\rho} \text{ and } [\![t_2]\!]_{X,\rho} \text{ are identical} \\ false & \text{otherwise} \end{cases}$$

Note that eq_X is non-strict and non-monotonic (and, therefore, non-continuous): for example, $\bot\ eq_X\ \bot = true$, while $\bot\ eq_X\ \xi$ = false, for ξ different from $\bot$.

On the other hand the symbol $\equiv$, having to represent a sort of semidecidable test of equality, must be given a continuous interpretation. Hence the largest set on which strict equality can be true is the set of algebraic maximal elements. Maximality ensures that equal elements cannot become distinct by adding more information. Algebraicity guarantees that the comparison can be done in a finite time (by exploring a finite amount of information). A possible solution is the assignment of a fixed interpretation of $\equiv$ which satisfies this requirement, for example:

$$[\![t_1 \equiv t_2]\!]_{X,\rho} = \begin{cases} true & \text{if } [\![t_1]\!]_{X,\rho} \text{ and } [\![t_2]\!]_{X,\rho} \text{ are algebraic, maximal and identical} \\ false & \text{if } [\![t_1]\!]_{X,\rho} \text{ and } [\![t_2]\!]_{X,\rho} \text{ are algebraic and maximal, but not identical} \\ \bot_{LL} & \text{otherwise} \end{cases}$$

Alternatively, we can consider $\equiv$ as an ordinary predicate (whose interpretations must be continuous by definition) and axiomatize its *truth*, by adding to the program the clauses:

$$d(x_1, ..., x_m) \equiv d(y_1, ..., y_m) \leftarrow x_1 \equiv y_1, ..., x_m \equiv y_m \qquad (m \geq 0)$$

for each constructor symbol d (this alternative will be the one choosen for implementing K-LEAF). Note that the two ways of defining $\equiv$ are equivalent, relatively to the set in which the value of strict-equality is *true*.

The meaning of non-atomic formulas is defined as follow:

$$[\![B_1, ..., B_n]\!]_{X,\rho} = [\![B_1]\!]_{X,\rho}\ and\ ...\ and\ [\![B_n]\!]_{X,\rho}$$

$$[\![A \leftarrow B_1, ..., B_n]\!]_{X,\rho} = [\![A]\!]_{X,\rho} \Leftarrow [\![B_1, ..., B_n]\!]_{X,\rho}$$

$$[\![\leftarrow B_1, ..., B_n]\!]_{X,\rho} = false \Leftarrow [\![B_1, ..., B_n]\!]_{X,\rho}$$

where *and* and $\Leftarrow$ are continuous extensions of the standard conjunction and implication [14].

Definitions. Let W be a set of Kernel or flat LEAF (program and goal) clauses. A *model* of W is any interpretation $M = ((X, \leq), [\![\,]\!])$ such that, for every clause cl in W and for every environment ρ, $[\![cl]\!]_{X,\rho} = true$. W is *consistent* if it has a model. A conjunction of atomic formulae, G, is true in M iff for every environment ρ $[\![G]\!]_{X,\rho} = true$. A conjunction of atomic formulae, G, is *valid* with respect to W, or is a *logical consequence* of W, iff it is true in every model of W.

A special class of interpretations is represented by Herbrand interpretations, which are based on a purely syntactical domain, the Herbrand universe. In our case the Herbrand universe is simply the set of the ground data terms, ground data partial terms and ground data infinite terms, i.e. the set of (possibly infinite) terms that can be built by means of the (constants and)

constructors of the language and of the new constant $\perp$. Infinite terms may be defined in the standard way as infinite (finite-branching) trees. The ordering is the usual approximation order, which correctly gives to the above set the structure of an algebraic CPO [14].

A *Herbrand model* for W is any Herbrand interpretation which is a model. In the following, HI will denote the set of Herbrand interpretations, while HM will denote the set of Herbrand models.

Let us point out the meaning of equality and strict equality on Herbrand interpretations. Equality simply is syntactic identity. To see the meaning of strict equality, note that in the CPO $(CU, \leq)$ maximal algebraic elements are the data terms which contain no occurrences of $\perp$. Hence strict equality is the syntactic identity only on the subset of the Herbrand Universe which is isomorphic to the ordinary "data-term" Herbrand Universe.

Consider the set of functions from a poset $(X, \leq)$ into a poset $(Y, \leq)$. This set is naturally ordered by the relation $g \leq g'$ iff $\forall x \in X \ g(x) \leq g'(x)$. The minimal function (Ω) is the function which maps every element of X in the bottom element of Y $(\forall x \in X \ \ \Omega(x) = \perp_Y)$.
This functional ordering induces an ordering on Herbrand interpretations:

$$\text{for } I, I' \in HI, \ \ I \leq I' \text{ iff } \forall f \in F \,.\, \forall p \in P. \quad [\![f]\!]_I \leq [\![f]\!]_{I'} \text{ and } [\![p]\!]_I \leq [\![p]\!]_{I'}.$$

The interpretation I_0,which maps in Ω every function and predicate symbol, is the minimal element of HI. Also $(HI, \leq)$ is a CPO, as proved in [14].

Herbrand models can be proved to keep, also in this kind of logics, the special role they have in standard logic. Namely, if W is a set of K-LEAF (program and goal) clauses, it is consistent iff it has a Herbrand model. Moreover, the lub of the Herbrand models of W is a Herbrand model, hence (if it is consistent) W has the *minimal Herbrand model*. Therefore a conjunction of ground atoms is valid in W iff it is true in the minimal Herbrand model of W.

It is possible to extend the standard transformation on Herbrand interpretations (used to define fixpoint semantics) so as to preserve monotonicity and continuity. Standard results still hold. Namely, the *least (minimal) fixed point* of the extended transformation T is equal to the minimal Herbrand model and can be effectively computed as the lub of the chain I_0,, $T(I_0,)$, $T^2(I_0,)$, ...

3.2. The execution of K-LEAF programs

The computational methods that have been proposed for the execution of languages based on Horn clause logic with equality are, in general, linear refinements of resolution and completion (i.e. SLD-resolution and narrowing, respectively). Among them we find conditional narrowing [10, 11] and SLDE-resolution (i.e. SLD-resolution with syntactic unification replaced by a E-unification [15, 26, 22]).

The technique we have chosen in the project is *flat SLD-resolution*, i.e. SLD-resolution on *flattened* programs augmented with the clause $x=x$. The flattening tranformation consists in eliminating functional composition by recursively replacing a term $f(t1,...,tn)$ with a new variable v and adding the functional atom $f(t1,...,tn)=v$ to the body. The original idea, in the theorem-proving domain, probably traces back to [9], while in the area of logic and functional

programming it was first proposed in [1, 27].

SLD-resolution on flat programs seems to be more adequate than narrowing, because:

- SLD-resolution was shown to be equivalent to "refined" narrowing [6], with a considerable gain in efficiency with respect to "ordinary" narrowing (elimination of redundant solutions and, more generally, reduction of the search space);
- the full (relational + functional) language can be supported by a single inference mechanism;
- conditional equations can easily be handled, without need of extensions;
- sharing of subexpressions deriving from a common expression is obtained for free.

The flattening algorithm for K-LEAF is similar to the one described in [2], with strict-equality atoms handled as ordinary predicates.

For instance, the K-LEAF program in Example 5 is flattened into:

```
samefringe(t1,t2) :-  fringe(t1)=v1, fringe(t2)=v2,v1 ≡ v2.
fringe(t) = v :- fringe1(t,nil)=v.
fringe1(leaf(x),continuation) = cons(x,continuation).
fringe1(tree(tl,tr),continuation) = v :- fringe1(tr,continuation)=v1,fringe1(tl,v1)=v2.
```

The flattening transformation is correct because an original K-LEAF program and its flat form have the same Herbrand models - where the notion of model for flattened programs is a trivial extension of the corresponding K-LEAF notion [14].

An objection which has been raised to this approach concerns the presumed loss of the producer-consumer information contained in the functional notation. On the contrary, this information is still implicitly present in the flat form, and can be exploited by the selection strategy.

A selection rule corresponding in the unflattened program to an innermost rule can be easily implemented through the usual leftmost selection rule of Prolog [6], provided the flat literals are put in the right order by the flattening procedure. This strategy has however a serious drawback in the unlimited possibility of resolving functional atoms with $x=x$, which results in a large amount of useless computation. The elimination of the reflexive clause causes the loss of completeness, unless functions are constrained to be everywhere defined, as in [11]. The problem can be overcome by noting that the resolution of a functional atom $t=z$ with $x=x$ is only useful when the resolutions of the atoms in whose arguments z occurs, bind z to variables (i.e. they do not require a value for z). This is in general not the case, and it cannot be determined statically.

The detection of this situation requires an *outermost* strategy, analogous to lazy evaluation in functional programming, which reduces a functional atom only when its output variable would be bound to a non-variable term by resolution of a consumer atom. Resolution with $x=x$ can then be profitably applied to the functional atoms whose output variables do not occur elsewhere in the goal, and may therefore be implemented as an *elimination rule* (which can also be viewed as an explicit garbage collection step). Moreover, when, in the resolution of an atom, unification attempts to instantiate a variable produced by another (functional) atom, resolution of the current atom is suspended and resolution of the producer is tried instead. The suspended goal is resumed after (one step of) resolution of (all) the activated functional atom(s).

Example 6.

Consider the program (which is already in flat form):

1) p(x,y) :- q(x,x).

2) q(a,x).

3) f(b) = a.

where a and b are constructors.

Let the goal be

?- p(f(x),g(x)). flattened into: ?- p(v1,v2),f(x)=v1,g(x)=v2.

The only outermost atom is *p(v1,v2)*, which is resolved with (1):

?- q(v1,v1),f(x)=v1,g(x)=v2.

The functional atom *g(x)=v2* can be eliminated, as the produced variable *v2* does not appear elsewhere in the goal:

?- q(v1,v1),f(x)=v1.

Resolution of *q(v1,v1)* is suspended, since (2) would bind the variable *v1*, produced by the other atom, to the non-variable term *a*. Resolution of the producer is executed instead: ?- q(a,a).

Now resolution of *q(a,a)* (i.e. *q(v1,v1)* with *v1* bound to *a*) with the clause (2) can be resumed. The goal thus succeeds with computed answer *{x:=b}*.

The strict-equality atoms can in principle be handled through their defining clauses like any other user-defined predicate. However, resolution with the $\equiv$-clauses can give rise to infinite branches of computation if atoms of the form $x \equiv y$, with x and y non-produced, are present. In this case the fake clause $x \equiv x$ is instead applied.

The outermost strategy, unlike the innermost case, cannot be implemented by means of a trivial compilation, because the atom selection order is not known statically, but can only be established at runtime. A more complex control of the computation than the one needed for Prolog is therefore required. While the selection order of the relational atoms is immaterial, the choice of the functional atoms to be resolved must be performed within the unification algorithm. The efficiency of the strategy is thus related to the efficiency in recognizing the produced variables and in finding their producers.

The complete definition of the outermost strategy is described in [7] and in [14], where it is proved correct and complete with respect to the declarative semantics.

As the design of an efficient sequential model is a preliminary step to any parallel implementation, we are now developing a K-LEAF abstract (sequential) machine. It consists in a modification of the Warren Abstract Machine (WAM) [30] where, to implement the outermost strategy, the unification instructions are changed as follows:

- functional atoms are represented as terms stored in the heap;
- to represent produced variables, a new kind of term is introduced which links the variable to its producer;
- unification instructions collect all the (terms denoting the) functional atoms that produce

the variables bound by the unification: these atoms are then resolved before body atoms.

Storing functional atoms as terms on the heap requires, to start their resolution, an efficient implementation of a meta-predicate similar to Prolog's *call* .

We have developed in C-Prolog a quite efficient "compiled-emulated" executor of K-LEAF where the unification instructions are emulated by Prolog predicates. Its natural evolution, planned for the next years, will be the implementation of the abstract parallel computational model described in [14].

4. Mapping Higher-Order L+F to First-Order L+F

The approach consists in taking an HCL+E axiomatization of the higher order language and in trying to efficiently execute this axiomatization, possibly specialized to the particular computational needs required by the context of application. A technique for introducing HO curried functions in the logic programming framework was first proposed by Warren [29]. He suggested to this end the definition and use of an *apply* predicate in Prolog programs. For example, the HO function *twice* would be written

```
apply(twice,X,twice(X)).
apply(twice(X),Y,R) :- apply(X,Y,R1), apply(X,R1,R).
```

To Warren's mind, the burden of building these clauses was to be left to the user. In our approach, on the other hand, they are automatically obtained by *partial evaluation* of the user-supplied function with respect to an axiomatization of lambda-calculus. This means that a procedure for deriving theorems from the axioms is used to reduce the application of a user function to a symbolic constant (i.e. to prove some universal properties of the function).

For example, with the function λ(x,λ(y,x+y)) we obtain:

(1) λ(x,λ(y,x+y))@A => λ(y,A+y)

and

(2) λ(y,A+y)@B => A+B

A "compiled" version of the function merely consists of the proved theorems (1) and (2). Observe that capital letters denote logical variables which are universally quantified when (1) and (2) are asserted in a logic database. From a "functional" viewpoint, the term *λ(y,A+y)* , produced by the first application, is a *closure* where *λ(y,_+y)* is the text component (which could be substituted by a new constructor) while A is the environment component to be kept for further applications. At least for the sequential case, there is a strong analogy between the traditional implementation of functional programs with closures built on the heap, and the most standard computational model for logic programming, namely the Warren Abstract Machine (WAM) [30], where the structure copying mechanism causes terms like the rightandside of (1) (an argument of the reduction relation) to be copied on the heap.

This approach was argued by Warren to be reasonably efficient in a Prolog environment equipped with the capability of indexing on the first argument of predicates, e.g. a standard WAM

with *switch-on-term/constant/structure* instructions which, at every call of the *apply* predicate, perform a hashed search of the function definition. Optimized specific handling of *apply* could be obtained with a slight modification of the WAM: the function definition could be entered immediately through a fast indirect step, where the address of the function body to be executed is fetched in a special field of the function name which contains the index of that name in the scope of the *apply* predicate.

We merely reobtained in a logic programming framework what in the functional programming community is called *lambda-lifting* [20], a technique used to find universal consequences of the reduction relation on a specific program. It basically consists in transforming a nested lambda-term into a set of *flat* rewrite rules, (or *super-combinators*): these rules are equivalent, with respect to reduction, to our *apply* clauses.

To produce by partial evaluation a "compiled" code like the one shown above, the interpreter of the HO language could be built, in principle, either in logic-programming style or in equational style. K-LEAF provides both alternatives. However, since for an interpreter of lambda-calculus the possibility of a leftmost strategy at the HO level must be guaranteed, the availability of a leftmost strategy already at the lower level allows a more compact and efficient axiomatization [O'Donnell85]. To this end, the so-called *micro-lambda-calculus* [21] has been adopted, in this first experimental phase, as the equational theory to be "implemented" in (the equational part of) our FO language. A variant which seems to have the normalization property, derived by the idea in [25], could be the following:

$\lambda X.X \mathbin{@} G = G \;:-\; \lambda var(X).$

$\lambda X.X \mathbin{@@} G = G \;:-\; \lambda var(X).$

$\lambda X.Y \mathbin{@} G = Y \;:-\; \lambda var(X), \lambda var(Y), X \neq Y.$

$\lambda X.Y \mathbin{@@} G = Y \;:-\; \lambda var(X), \lambda var(Y), X \neq Y.$

$\lambda X.E@F \mathbin{@} G = (\lambda X.E \mathbin{@} G)@(\lambda X.F \mathbin{@} G) \;:-\; \lambda var(X).$

$\lambda X.E@F \mathbin{@@} G = (\lambda X.E \mathbin{@} G)@(\lambda X.F \mathbin{@} G) \;:-\; \lambda var(X).$

$\lambda X.\ \lambda Y.B \mathbin{@} G = \lambda Y.\ (\lambda X.B \mathbin{@@} G) \;:-\; \lambda var(X),\ \lambda var(Y), X \neq Y,\ Y \notin varfree(G).$

The compilation we could obtain by such a method looks like the following:

twice@X = twice1(X)

twice1(X)@Y = X@(X@Y)

or alternatively

twice@X = twice@'X

twice@'X@Y = X@(X@Y)

where @' is the "constructor" version of the function @, denoting the *closure*. These are correct K-LEAF programs.

4.1 Unification

If the general axiomatization of lambda-calculus is not kept at "run-time", and the compiled code is executed alone, unification at the FO level has a limited capability of solving equations in functional variables, which can only be instantiated to functional forms (functions) present in the original program. For example, if the simple definition plus1@X = s(X) is added to the ones above, the execution of the goal *F@a=s(s(a))* yields the substitution *F=twice@'plus1*; on the other hand, the solution *λ(x,s(s(x)))* is not found out because this term does not belong to the Herbrand space of the "compiled" program.

This kind of invertibility can be achieved with a good efficiency, on the basis of the present K-LEAF implementation. The user interested in *synthesizing* programs should introduce a library of basic functions over which the search has to be performed. An external methodology could enforce the introduction in the library of second-order patterns which guarantee some correctness properties (well-typing in a broad sense). Suppose, for example, that a specification of an unknown recursive function *F* requires *F@3=6*. We could try to see whether, given a set of primitive functions, a particular recursive scheme *s* :

s(Arg,Test,End,F1,F2) = if Test then End
else F1@s(F2@Arg,Test,End,F1,F2)

fits our I/O specification. This amounts to requesting the evaluation of the term

?- s(3,Test,End,F1,F2) = 6

which, in presence of a library of "fully" axiomatized integer functions like *plus*, *times*, etc., produces the solution

Test = eq0, End = 0, F1 = plus1, F2 = sub1

corresponding to the function computing the sum from N downto 0, and the solution

Test = eq0, End = 1, F1 = times, F2 =sub1

corresponding to the factorial function.

In conclusion, this approach amounts to translating the higher order program into the set of equations arising from the partial evaluation of an equational axiomatization of beta-reduction, and solutions of functional equations can be found only in lambda-terms already present in the program.

A natural generalization of the method consists in using the axiomatization itself and the FO-level inference method (roughly a form of narrowing) to obtain more general solutions for functional equations. In other words, equations between lambda-terms can be solved by executing a general E-unification algorithm on a particular equational theory for the lambda-calculus.

Because the problem of unification in lambda-calculus has been proven to be only semidecidable even in the typed case [19,16], it does not follow that an algorithm which eventually finds unifiers, if they exist, is useless. Application domains like program transformation or program synthesis could supply sufficient edge conditions so as to make undecidabilty relatively irrelevant for the practical usage.

We have not taken into consideration specialized unification algorithms, like the one in [19], recently chosen by [23] in the integrated L+F language lambda-Prolog as the basic unification

algorithm, to be used only when really needed. Our goal, at this stage of the research, rather consists in trying to identify one simple execution mechanism for both logic and functional programming.

In this perspective, our interest has concentrated on the *narrowing* technique, which can be efficiently supported by a resolution machine [6]. Our present efforts are aimed at finding out which extra-features should be added to standard narrowing procedures in order to be able to deal with non-terminating systems like those for lambda-calculus, which do not satisfy the constraints of the language K-LEAF (non-superposition, distinction between functions and constructors, treatment of the occur-check). Recently [31], narrowing has been proved complete for a class of non-terminating systems satisfying the "non-repetition" constraint, which means that a same rule cannot be indefinitely applied to E-equivalent terms. Unfortunately this result doesn't apply to lambda-calculus.

Some form of the so-called *narrowing-on-variables*, which is forbidden in standard algorithms, seems to be mandatory in the presence of rules like $f = c(f)$, if the execution of a goal like $?\text{-}X = c(X)$.must be able to produce the solution $X := f$. Following the idea suggested in [18], we are now experimenting with a modified narrowing algorithm where narrowing-on-variables is only tried on occur-check failure. More precisely, when a goal $t(..X...)=t(..f(..X..)...)$ fails for occur-check and $f(..X..)$ is no more narrowable, a rule of the form $l \rightarrow f(...)$ can be applied to narrow the variable X, and the goal becomes $t(..f(...)...) = t(..f(..X..)...)$. With this modification, the narrowing algorithm is able to solve equations like

```
        F = z@F              (i.e. find a fixpoint of z)
        F:=lambda(x,z@(x@x))@lambda(x,z@(x@x))
or      Y@f = f@(Y@f)        (i.e. find a general fixpoint combinator)
        Y:=lambda(z,lambda(x,z@(x@x))@lambda(x,z@(x@x)))
```

As a matter of fact, these solutions have been obtained by "driving" the narrowing process so as to narrow only the "interesting" subterms, and to disregard those which could lead to infinite failing computations.

One important source of infinite branches is the axiomatization of lambda-variables. If conditions of rules (declaring that the first arguments of lambda's must be lambda-variables) are evaluted before the narrowing step, (names of) lambda variables are actually generated, so introducing an infinite nondeterminism. This problem can be solved by a more complex narrowing strategy, where some conditions are carried over unresolved as constraints, and sometimes checked for satisfiability. In this case we may obtain solutions equipped with a set of satisfiable constraints. Of course falsity of constraints has to be computed as soon as possible. Some special meta-level primitives, hidden to the user, could be profitably embody some special cases of induction. Consider, for example, the $dif(X,X)$ test (where X is a logical variable), which yields *false* for every assignment of the variable X. The satisfiability test can be exemplified by the following Prolog clause

```
dif(X,Y) = false :- var(X),var(Y),X==Y.
```

where, in order to reduce the constraint as soon as possible, the special primitives *var* and == are used, in a sound way.

4.2 Alfa and eta conversions

If we have an algorithm for confluent theories, eta-conversion does not pose particular problems. It can be added as a rewrite rule:

$\lambda(X,(M@X)) = M$:- $X \notin$ varfree(M).

As for alfa-conversion, in principle it could be introduced as a rewrite rule (actually an expansion rule)

$\lambda(X,B) = \lambda(Z,lambda(X,B)@Z)$:- $Z \notin$ varfree(B).

which executed by narrowing can prove alfa-equality with an infinite number of failing reductions (expansions). However, to cope with the alfa-equality in a "sensible" way, it is wise to embed it in the syntactic unification. The algorithm thus becomes a beta-eta-narrowing modulo alfa. If we represent syntactic equality by the term *eq(X,Y)* the alfa-equality can be expressed as follows (but syntactic unification is implemented at a machine level):

eq(lambda(X,B),lambda(Y,B1) = eq(lambda(X,B)@Z,lambda(Y,B1)@Z) :-

$Z \notin$ varfree(B),$Z \notin$ varfree(B1).

4.3 Impacts on the First-Order language

The scenario presented above, which has to be considered a research theme rather than a set of fully achieved results (in the sense that the completeness of the approach has not yet been proved), involves several assumptions on the underlying FO language. Some modifications and relaxations of constraints will be needed in K-LEAF, to make it able to directly support the kind of narrowing sketched above, necessary for a complete treatment of lambda-calculus. Among these features we recall: less strong distinction between functions and constructors, relaxation of the non-ambiguity constraint, introduction of the occur-check, strategies for carrying over unresolved constraints, ways of connecting with special algorithms for constraint satisfiability and (for more general purposes) with special unification algorithms.

4.4 Efficiency

So far the only figures about efficiency of functional programs compiled in a logical FO language are those obtained with the prototype language IDEAL, which is, as already mentioned, implemented in Prolog, following the original Warren's idea. We report here the timings (in seconds) relative to a simple benchmark (the computation of the list of permutations of a

six-elements list) which exhibits a reasonable amount of closure construction, the distinctive feature for this kind of comparison. Three languages are compared, running on VAX780: Digital's Common Lisp and INRIA's LeLisp (with lexical scoping) both in their interpreted and compiled versions, and IDEAL, whose code is run by the C-Prolog interpreter and by Quintus Prolog.

```
map(F,[],C) = C,
map(F,[X|L],C) = [F@X|map(F,L,C)].
insert(A,[]) = [],
insert(A,[[]|L]) = [[A]],
insert(A,[L|L1]) =  [[A|L]|map(lambda(E,[hd(L)|E],
                        insert(A,[tl(L)]),
                        insert(A,L1))].
perm([]) = [[]],
perm([A|L]) = insert(A,perm(L)).

perm([1,2,3,4,5,6]) = ?.
```

	Compiled	Interpreted
IDEAL (C-Prolog)		4.6
IDEAL (Quintus Prolog)	1.9	22
VAX Common Lisp	1.5	32
LeLisp	2.4	4.3

These figures, though having to be confirmed by a larger set of benchmarks, are quite encouraging. Similarly satisfactory results have been shown in [17], where functional programs compiled into Prolog have been compared with the equational interpreter of [24].

5. Conclusions

Though the overall ESPRIT Project N.415 is on parallel architectures, and in this framework also the subproject D will produce as final result a parallel virtual machine, much of the work of the first two years concentrated, as scheduled, on the design of the L+F language, which was not in existence at the start. The result of the effort has been described in this paper, an can be judged quite satisfactory, even in comparison with what has been achieved elsewhere in the same domain.

While the choice of HO language as the user *ideal* language was mainly dictated by the need of powerful programming capabilities, the design of a new FO language to replace Prolog as intermediate compilation language was forced by the need of a semantic characterization, to start with, of the goals computable at the first order.

As regards parallelism, we are considering the applicability to the L+F language of the annotation scheme and computational model devised in [12], and the mapping of these on the machine described in [8].

Acknowledgement

This work has been partially sponsored by EEC under ESPRIT Project 415 "Parallel Architectures and Languages for Advanced Information Processing - a VLSI-directed approach".

References

[1] **R. Barbuti, M. Bellia, G. Levi and M. Martelli,** On the integration of logic programming and functional programming, Proc. 1984 Symp. on Logic Programming (IEEE Comp. Society Press, 1985), 160-166.

[2] **R. Barbuti, M. Bellia, G. Levi and M. Martelli,** LEAF: A language which integrates logic, equations and functions, in Logic Programming: Functions, Relations and Equations, D. DeGroot and G. Lindstrom, Eds. (Prentice-Hall, 1986), 201-238.

[3] **M. Bellia and G. Levi,** The relation between logic and functional languages: A survey, Journal of Logic Programming 3 (1986),217-236 .

[4] **P.G. Bosco and E. Giovannetti,** IDEAL: An Ideal DEductive Applicative Language, Proc. 1986 Symp. on Logic Programming (IEEE Comp. Society Press, 1986), 89-94.

[5] **P.G. Bosco, E. Giovannetti and C. Moiso,** A completeness result for a semantic unification algorithm based on conditional narrowing, to appear in Proc. Foundations of Logic and Functional Programming (Trento 15-19 December 1986).

[6] **P.G. Bosco, E. Giovannetti and C. Moiso,** Refined strategies for semantic unification, Proc. TAPSOFT '87, LNCS 250 (Springer-Verlag, 1987), 276-290.

[7] **P.G. Bosco, E. Giovannetti, G. Levi, C. Moiso and C. Palamidessi,** A complete semantic characterization of K-LEAF, a logic language with partial functions, Proc. 1987 Symp. on Logic Programming (IEEE Comp. Society Press, 1987).

[8] **P.G. Bosco, E. Giachin, G. Giandonato, G. Martinengo and C. Rullent,** A parallel architecture for signal understanding through inference on uncertain data, Proc. PARLE Conference, LNCS 258 (Springer-Verlag,1987), 86-102.

[9] **D. Brand,** Proving theorems with the modification method, SIAM J. Comput. 4 (1975), 412-430.

[10] **N. Dershowitz and D.A. Plaisted,** Logic Programming cum Applicative Programming, Proc. 1985 Symp. on Logic Programming (IEEE Comp. Society Press, 1985), 54-66.

[11] **L. Fribourg,** SLOG: A logic programming language interpreter based on clausal superposition and rewriting, Proc. 1985 Symp. on Logic Programming (IEEE Comp. Society Press, 1985), 172-184.

[12] **G. Giandonato and G. Sofi,** Parallelizing Prolog-based inference engines, ESPRIT Project 26, T4.3 Techn. Rep. (Sept. 1986).

[13] **E. Giovannetti and C. Moiso,** Some aspects of the integration between logic programming and functional programming, to appear in Proc. of AIMSA '86 (North-Holland).

[14] **E.Giovannetti, G. Levi, C. Moiso and C. Palamidessi,** Kernel LEAF: an experimental logic plus functional language - its syntax, semantics and computational model, ESPRIT Project 415, Second year report (1986).

[15] **J.A. Goguen and J. Meseguer,** Equality, types and generic modules for logic programming, in Logic Programming: Functions, Relations and Equations, D. DeGroot and G. Lindstrom, Eds. (Prentice-Hall, 1986), 295-364.

[16] **W. Goldfarb,** The undecidability of the second order unification problem, Theoretical Computer Science 13 (1981), 225-230.

[17] **J. Heering and P. Klint,** The efficiency of the equation interpreter compared with the

UNH Prolog interpreter, SIGPLAN Notices 21, n. 2 (ACM, 1986). 18-21.

[18] **S.Holldöbler**, A Unification Algorithm for Confluent Theories, Personal Communication (1986).

[19] **G. Huet**, Resolution d'equations dans les langages d'ordre 1,2,...ω, These de Doctorat d'Etat, Universite' Paris VII (1976).

[20] **T. Johnsson**, Lambda-lifting: Transforming Programs to Recursive Equations, Proc. of Int. Conf. of Functional Programming Lnguages and Architectures, LNCS 201(Springer-Verlag,1985), 190-203.

[21] **J.W.Klop**, Term Rewriting Systems, Notes for the Summer Workshop on Reduction Machines (Ustica, 1985).

[22] **A. Martelli, C. Moiso and G.F. Rossi**, Lazy unification algorithms for canonical rewrite systems, to appear in Proc. of Colloquium on Resolution of Equations in Algebraic Structures, Lakeway, May 4-6 (Prentice-Hall).

[23] **D. Miller and G. Nadathur**, Higher-Order Logic Programming, Proc. of Third Int. Conf on Logic Programming, LNCS 225 (Springer-Verlag,1986), 448-462.

[24] **M. O'Donnell**, Equational Logic as a Programming Language, (M.I.T. Press, 1985), 54-62.

[25] **G. Revesz**, Axioms for the theory of Lambda-conversion, SIAM J. COMP. vol.14, n.2 (May 1985), 373-382.

[26] **P.A. Subrahmanyam and J.-H. You**, FUNLOG: A computational model integrating logic programming and functional programming, in Logic Programming: Functions, Relations and Equations, D. DeGroot and G. Lindstrom, Eds. (Prentice-Hall, 1986), 157-198.

[27] **H. Tamaki**, Semantics of a logic programming language with a reducibility predicate, Proc. 1984 Int. Symp. on Logic Programming (IEEE Comp. Society Press, 1984), 259-264.

[28] **D.A. Turner**, MIRANDA: a non-strict functional language with polymorphic types, Proc. of Int. Conf. of Functional Programming Lnguages and Architectures, LNCS 201(Springer-Verlag,1985), 1-16.

[29] **D. H. D. Warren**, Higher order extensions to Prolog. Are they needed?, Machine Intelligence 10 (Ellis Horwood, 1982), 441-454.

[30] **D. H. D. Warren**, An Abstract Prolog Instruction Set, Technical Note 309, SRI International (Oct.1983).

[31] **J.-H. You and P.A. Subrahmanyam**, E-unification algorithms for a class of confluent term rewriting systems, Proc. ICALP'86, LNCS 226 (Springer-Verlag, 1986), 454-463.

Project No. 818

DELTA - 4 : Definition and Design of an open Dependable Distributed computer system architecture

P. MARTIN

Bull SA
1 rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE

Tel.: (33) 76 39 76 13
Telex: 980648

1. INTRODUCTION

The overall aim of Delta-4 project (n°818) is the Definition and Design of an Open Dependable computer system architecture covering a wide application area, especially CIM and Office Systems. Three key quality attributes of the Delta-4 architecture are dependability, performance and software portability.

Following successful completion of the pilot phases of the ESPRIT Delta-4 and Concordia projects, the Commission of the European Communities has approved a large scale continuation project which combines the two. The project is continuing under the title of Delta-4; this second phase started on 1st March 1987 and will have a duration of two years and a budget of 8.8 MECU. The industrial partners of the present consortium are Bull (F) prime contractor, BASF (D), Ferranti (UK), Jeumont-Schneider (F) and Telettra (I). The research partners are IITB-FhG (D), FIRST-GMD (D), IEI-CNR (I), INESC (P), LAAS-CNRS (F), LGI (F), MARI (UK) and the University of Bologna (I).

2. MAIN OBJECTIVES

The Delta-4 project is targeted to provide a distributed environment for several application areas. These applications present generic aspects in terms of Information Processing, Communication Systems, Data base Systems, Object sharing, Man-machine Interface. Specific applications such as Computer Integrated Manufacturing, Process Control, Office System, Transport Information System, Banking System,..., increasingly require a fault-tolerant distributed support.

In the area of fault-tolerance and distributed systems, most of the existing products are of US origin. As examples Tandem, Stratus and August for fault-tolerance, Locus, Apollo and Sun for distributed systems. The fragmentation of Community manufacturers' activities has not produced recognised products in this area.

Most of the existing solutions do not offer a sufficient openness by the fact that they are limited to specific applications, and that they are proprietary, making difficult, and even impossible, use of these solutions for other existing computer systems. The environment provided by the Delta-4 project is OPEN, in the sense that it allows the use of existing proprietary computer systems, it accomodates the connection of heterogeneous equipment and it is capable of co-existing with, and interworking with, ISO standards conforming to the OSI model.

The main objective of Delta-4 is to elaborate an open european solution, that improves the current state-of-the-art in terms of fault-tolerance and distributed systems, and that allows european companies to offer competitive products on the world-wide market.

In order to achieve this objective, the Delta-4 project has built up a european association of users, computer systems manufacturers and research organisations, all of these partners having an important background in the above-mentioned areas.

The delta-4 project does not content with defining new concepts, but also intends to apply and to demonstrate them. The demonstration prototypes within the project have been formulated to show, at regular intervals, tangible advances in a progressive manner which can be readily exploited and engineered into commercial products by european industry. The distributed architecture is a living architecture accomodating technological evolution by means of continuous prototyping. A first demonstration has already been provided using Phase 1 prototypes.

3. OVERALL APPROACH TO DEPENDABILITY AND DISTRIBUTION

The major objective of the Delta-4 architecture is to enable modular configurations of systems offering a range of fault-tolerance and performance in an open distributed environment.

DEPENDABILITY is the quality of the delivered service such that reliance can be justifiably placed on the service. It includes reliability, safety and security. Dependability avoids system failures that may be costly in loss of human lives, productivity, custom and privacy. Dependability is procured within Delta-4 by fault-tolerance techniques. Fault-tolerance is the provision, by redanduncy, of proper service despite occurence of faults.

Distribution and fault-tolerance are tightly related. Should a single element of a distributed system fail, users expect at worst a slight degradation of the service that they are offered; distributed systems must thus at least have some built-in fault-tolerance. On the other hand, most fault-tolerant systems can, at some level, be seen as a distributed system due to their redundant processing resources.

The Delta-4 project pursues this tight relationship between distribution and fault-tolerance in order to offer an architecture that is both open and dependable. Applications supported by the Delta-4 architecture can be made incrementally fault-tolerant on a service-by-service basis. At system configuration time, the application designer can choose which services he wishes to make fault-tolerant and to which degree. Several techniques for fault-tolerance are supported; the application designer can thus choose,

according to the available resources, the technique that best suits his application and make the necessary trade-offs between fault-tolerance and performance.

The management of fault-tolerance and distribution will nevertheless remain invisible to the application programmers. This allows SOFTWARE PORTABILITY i.e. to re-use existing or part of existing software.

The Delta-4 architecture consists of a number of (heterogeneous) host computers interconnected by a dependable communication system, the application software being executed by the host computers.

One of the main dependability goals of Delta-4 is fault-tolerance with respect to hardware faults. Delta-4 achieves the tolerance of host hardware faults through the use of replicated software components executing on distinct hosts.

It is important in an open architecture using existing proprietary computer systems that the fault-tolerance mechanisms be capable of withstanding arbitrary host failure behaviours ("fail-uncontrolled" hosts). However, the architecture is also able to take benefit of "fail-silent" hosts, i.e. hosts possessing autonomous error-detection and confinement mechanisms that can be assumed to stop if they fail.

It is also intended to study new techniques for ensuring the confidentiality, authenticity and availability of information in high-security applications based on intrusion-tolerance (tolerance of intentional faults).

With regard to communication issues, Delta-4 selects among existing ISO services, those that are necessary, and extends these to take into account not yet satisfied needs, especially those related to fault-tolerance requirements. The Multicast Communication System (MCS) provides multi-end-point communications and is based on atomic multicast protocols, using logical designation. Multicasting is the basis for the management of replicated software components. The MCS uses standard Local Area Networks (DIS 8802.5 Token Ring, DIS 8802.4 Token Bus and ANSI X3 T9.5 FDDI). The atomic multicast protocols located at layer 2 of the OSI model extend the standard MAC (Medium Access Control) protocols, allowing MCS to co-exist and to interwork on the same LAN with ISO communication systems, especially with MAP for CIM communications. MCS enables portability of ISO applications by providing a superset of the ISO Common Application Service Elements (CASE).

Delta-4 is also contributing dependability concepts and techniques, through ECMA, to the new ISO upper layer architecture work item on Open Distributed Processing. The computational Reference Model which forms the basis of this work was promoted by the UK Alvey Advanced Network System Architecture Project - ANSA. Delta-4 is collaborating with this project in evaluating the relationship between the ODP Reference Model and dependability. We are doing this by implementing an instance of the proposed ODP Support Environment which will embody our dependability models. This instance, to be known as DELTASE, will be complemented by

language constructs for application binding and be directly supported by the underlying MCS.

The Delta-4 architecture provides a range of PERFORMANCE in terms of responsivity and respect of the various deadlines imposed by the different targeted applications.

4. MAIN ACHIEVEMENT OF PHASE 1 (March 86 - February 87)

Due to the reduced scale of the one year first phase of Delta-4 by comparison with the long-term objectives of the project, it was necessary to follow two different approaches:

- a top-down approach that consisted in undertaking research to identify the project requirements and to define appropriate concepts, methods and architecture in order to fulfil the main features of Delta-4, that are distribution, performance, dependability and openness

- an anticipative approach, whose goal was to develop some basic component prototypes of the Delta-4 system, which had been considered as of general interest in any case, i.e. whatever the results of the top-down approach were.

The main results of Delta-4 phase 1 were as follows :

- Delta-4 Overall System Specification; issued in August 1986 with the publication status open, this deliverable includes the requirement and objectives of the Delta-4 project, the dependability and performance concepts, an overview of the proposed architecture and the relationships to OSI model and standards.

- Dependability in Delta-4: concepts and techniques. Provided in February 1987 with the publication status restricted, this specification includes items such as:

    ° types of faults taken into account in Delta-4

    ° policies of fault-tolerance management

    ° mechanisms for effective error processing ; transaction concept

    ° computational model issues

    ° Communication-, Fault-tolerance- and Maintenance-Network Management

- Cluster Multicomputer Specification; issued in February 87 (publication status restricted), this specification includes the design concepts of the hardware and operating system

software for a high-performance and cost-effective Network Station to support the Delta-4 System architecture. This specification introduces items applicable to the whole Delta-4 architecture such as:

- an Applications Support Environment (DELTASE)
- a method of invoking remote operations (RSR: Remote Service Request).

Delta-4 Phase 1 provided three prototypes and the corresponding functional specifications:

- Trial implementation of Inter Process cooperation protocols; a software generation utility, interface generator, activator, and language and communication libraries, which activate servers and transfer data between client and server, have been implemented. The implementation is based on Modula 2 and UNIX and operates RSR.

- Real Time UNIX prototype; the extensions to UNIX System V.2 include priority and deadline scheduling and the associated new system calls, detection of missed deadlines, preemptive scheduling of high priority process with respect to lower priority processes, reduction of interrupt latency.

- Multicast Communication System prototype; this first prototype was based on an IEEE 802.5 Token Ring LAN. The generalisation of ISO standard point to point communications towards MCS multipoint communications induced the definition and implementation of adapted Session and Transport services and protocols, while the MAC services and protocols are an extension of DIS 8802.5 (Token Ring), offering an ascendent compatibility, i.e. providing both an ISO access and a multicast access.

5. FIRST DELTA-4 DEMONSTRATION

Delta-4 demonstrators are intended to be exhibited every year during the project life. The first demonstration was shown at the end of Phase 1; it demonstrated the three prototypes listed above.

This demonstration is divided in three separate parts:

a) The Real Time UNIX demonstration

This demonstration, on a Ferranti UNIMAX machine, shows extensions to UNIX System V to support real-time processes as well as standar UNIX facilities. Various mixes of real-time and time-sharing processes are shown competing for CPU time under a pre-emptive real-time scheduler which takes each process's priority and optional deadline into account.

b) The Remote Service Request (RSR) demonstration

This demonstration shows the prototype high-level language construct supporting RPC-like invocation of remote services in parallel with the continued execution of client processes. This first prototype implements the construct with pseudo-remote services: it is scheduled in the present phase of the project to be integrated with DELTASE and hence with the Multicast Communications System.

c) The Multicast Communication System (MCS) demonstration

This demonstration involves heterogeneous computer systems: one Ferranti UNIMAX machine, three Bull SPS7 machines and two IBM compatible PCs.

All of these equipment are connected to a single Token Ring LAN. The two PCs communicate through standard protocols, while the four other machines simultaneously use MCS.

The distributed demonstration application consists of:

- a file server that performs accesses to a file on behalf of client processes,

- one or several client processes requesting over the network read or write accesses to the file.

The file server may be replicated on different equipment; different scenarios are implemented according to the supposed fault-tolerance attributes of the stations ("fail-silent" or "fail-uncontrolled") and to the fault-tolerance attributes the user expects from the file server ("don't care", "error detection" or "service continuity").

This application wants to demonstrate that MCS enables:

- the communication between heterogeneous computer systems, co-existing on the same LAN with equipment using standard protocols (OPENNESS),

- to garantee the consistency of the file, whatever be the number of server copies or of client processes (ATOMICITY, CONSISTENCY of the file copies).

- to reach the dependability requirements of the user according to the dependability attributes of the stations (ERROR DETECTION, ERROR RECOVERY, DYNAMIC RECONFIGURATIONS).

The Delta-4 project is also presented at this ESPRIT Conference week 87 by means of this Phase 1 demonstration.

6. MAIN OBJECTIVES OF DELTA-4 PHASE 2 (March 87 - February 89)

During Phase 1, possible techniques for achieving dependable system operation in Open Systems were analysed. The analysis concluded that these techniques permitted systems to be built from a heterogeneous set of hosts, each of which may be fail-silent or fail-uncontrolled. It was shown that each technique could be supported by MCS.

One of these techniques, the Passive Replicate Model, has been pursued independently by the ESPRIT CONCORDIA project. The Delta-4 work has prototyped components which can support all techniques, but for analysis purposes has primarily concentrated on the other identified approaches. This complementarity has allowed the Delta-4 and Concordia partners to combine resources to undertake the second phase of work.

In phase 2, it is planned to develop sufficient prototype system elements to support the mounting of an industrial-scale demonstrator project under ESPRIT 2. In preparation for such a project, a major industrial user, BASF, has already joined the project to provide guidance on user requirements and to plan suitable applications which would permit the various prototypes to be integrated.

In parallel, work already started in phase 2 in order to validate design and protocols and to undertake the standardisation of components of the Delta-4 architecture by direct involvement on standardisation committees such as ECMA and IEEE.

The technical issues for the distributed architecture in Phase 2 are as follows:

- extensions to MCS including enhanced functionalities (inter-connection of LANs, CASE services), performance (real time services, design of custom VLSI integrating atomic multicasting on Token Ring) and versatility (through implementation of Token Bus and fibre optic Token Ring connections, redundant medium).

- a computational-model-related work which provides the consistent overall framework for the use of the dependability-related mechanisms, and also language support for a distributed Object Model and an open Application Support Environment,

- implementation of Communication, Fault-tolerance, and Maintenance Network Management functionalities,

- specification and prototyping of fail-silent controllers,

- analysis and specification of a technique based on intrusion

tolerance through the use of fragmentation-scattering in a file storage system,

- integration of all of these components in the demonstrators,

- specification and implementation of demonstration applications.

7. CONCLUSION

The project is active both in specifying generic distributed architectural features and in developing resulting prototypes. The presently designed environment for distributed applications could be completed in an ESPRIT 2 project, by a distributed system environment.

Active contribution to the standard committees will be developed, particularly for ODP (formerly known as DASE) at the ECMA committee; relationships with other european project(s) are intended to be established, in particular in the area of distributed architectures, and Office System and Computer Integrated Manufacturing applications. As an example, a cooperation with the UK Alvey Advanced Network System Architecture project has already started.

The importance of demonstrating the total architecture in an industrial environment is recognised and a further project is planned under ESPRIT2 in which large-scale pilot sites will be installed. Within this framework, it is envisaged to extend the present consortium to other users and to other computer systems manufacturers.

Project No. 967

PADMAVATI* : PARALLEL ASSOCIATIVE DEVELOPMENT MACHINE AS A VEHICLE FOR ARTIFICIAL INTELLIGENCE

by

Philippe ARSAC
Project Manager

THOMSON-CSF Division CIMSA-SINTRA
Parc d'Activités KLEBER
160, Boulevard de Valmy
B.P. 82
92704 COLOMBES

Abstract

This ESPRIT project, started on 24th of February 1986, is a cooperative effort between THOMSON-CSF Division CIMSA-SINTRA (prime contractor), CSELT (Italy), and GEC (United Kingdom). Are involved as subcontractors : THOMSON-CSF LER (France), NON STANDARD LOGICS (NSL France), PROLOGIA (France), UNIVERSITA DI TORINO (Italy), POLITECHNICO DI TORINO (Italy).

Its objectives are to provide a software and hardware mock-up and give way to symbolic architecture concepts achieving high performances validation. Domains investigated are parallelism in LISP and PROLOG, associative devices at different hierarchical levels, preliminary studies in knowledge based system connection and in symbolic and numerical integration. First year was a system definition phase, middle stage will be hardware and software integration. During the last year we will implement applications in the fields of image recognition (LER), speech understanding (CSELT), natural language (GEC), and study interactions with PROLOG III.

PADMAVATI is a MIMD symbolic dorsal architecture connected with a MicroVax host. For performance and flexibility compromises : processing is based on the Transputer, a Delta Network provides non local communications, and the links connecting the Transputers into rings convey the local communications. We have distinguished for PROLOG : Dispatching Processors (DPs) and Processing Elements (PEs). With the implementation of PROLOG on a (DP) and a (PE) we study microparallelism in sequencial PROLOG interpreters and validate Associative Devices concepts. Studies in parallel PROLOG led us to the definition of two different PROLOG models. One is close to a dynamic sort of "Delta Prolog", the other is annotated and uses "Intelligent Backtracking". Studies in parallel LISP led us to provide the parallel machine with an implementation of a standard LISP system (Le_Lisp), extended with facilities for explicit concurrency. Asynchronous communications, debbuging facilities, host interfacing and loose communication with PROLOG, are ensured by the run time support.

*This research is 50 % founded by the European Economic Community (EEC) under contracts for ESPRIT Project 1219 (967).

1 - INTRODUCTION

Since the announcement of the Japanese Fifth Generation project, researchers all over the world have been working even harder towards results compatible with the industrial demand at a middle range date.

However very few projects have focused simultaneously on the hardware level, the parallel computational model, the application level ; and made finally all work together in real scale.

This is the goal of the ESPRIT Project n° 1219 (967) "PADMAVATI", the mean being the partners having complementary expertise in order to provide a near term oriented mock-up. It is designed to support parallel LISP and parallel PROLOG in a efficient way.

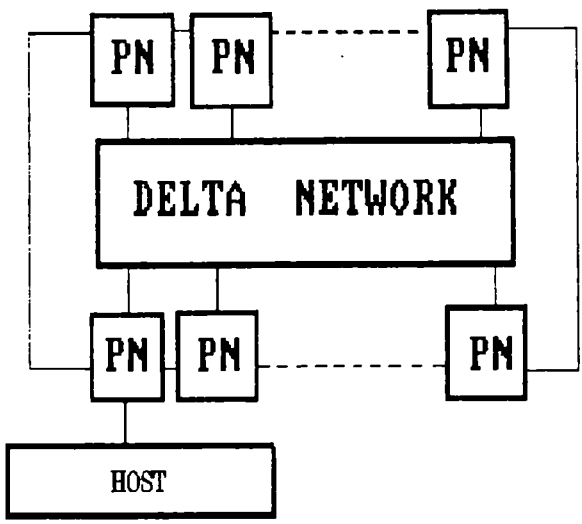
At the hardware level, our research had to provide enough performance possibilities, keeping flexibility to support different explicit/implicit computational models.

The main difficulty is that when a sufficient grain of parallelism is sought after (100 processors or above), physical, language and algorithmic levels interact to a considerable extent.

Because of the considerations reported in 2.1, we retained for the basis of our architecture the one defined in project ESPRIT n° 26 (fig. 1). N Transputer processors, up to 128, are interconnected by :

- a packet switched network of the Delta topology,
- a local network constituted of Transputer links.

Figure 1 : PADMAVATI basic architecture



This highly parallel machine was adequate for investigation of various computation schemes. However we wanted to investigate speed-up of each node as well as the best repartition of the work and data between them.

We thought associative devices would accelerate PROLOG whose key feature is unification, and accelerate LISP for functions using pattern matching. We remembered D.H.D. Warren's [14] dictum :

Unification = Pattern Matching + the Logical Variables.

We decided to incorporate such features in the architecture. The interesting point being that the speed-up of associative processors would add to the speed-up obtained by parallelism.

Exploration of the solutions and validation of the concepts have been achieved. They concern Hash Coding techniques, Content Adressable Memories (CAM) and Filtering Operators as precised further. We will measure the improvements in real context in the next stages of the project.

A workable model of LISP has been defined, suitable for continuous speech understanding.

Several PROLOG computational models have been studied. This study was at language level and also about the repartition of the data and programs between the processors.

Applications and the related algorithms are studied in this project : image recognition, speech understanding, natural language. Useful informations about continuous speech understanding can be found in [3]. However the application parts will not be related in this paper.

We will discuss successively the main conclusions drawn at this stage at the hardware and basic software level, on different points above-mentioned. Our report is splitted in two parts : architecture and software. It is only a convenient way of presenting things and does not reflect our thought process.

2. HARDWARE ARCHITECTURE

2.1. Basic architecture

We had in mind future VLSI implementation of the machine. This led us to reject solutions based on a centralized shared memory, incompatible with that approach.

Our basic choice was to use a concept of global distributed storage, already investigated in other contexts, at first with the CM*, then followed by Darlington's ALICE, Keller's REDIFLOW, the BBN Butterfly and the RP3.

In our MIMD model, each node is seen as a Processing Node (PN) accessing locally to a page of the global memory space, and accessing through remote pointers to objects located in other PN's.

We thought that a packet-switched network, belonging to the family of Delta network, was the ideal interconnection structure for a system being both highly parallel and asynchronous.

At the PE level fast switching capabilities were required to tolerate long latency times in accessing remote memory.

A sufficiently fast context switching of 1/µs (1 order of magnitude faster than other microprocessors), the OCCAM language providing synchronisation at

the system level, led us to the TRANSPUTER choice.

This basic architecture is summarized in figure 1.

Details about the 2 types of connections between the nodes and their motivations are related in detail in [3].

2.2. Hardware enhancements for PROLOG

Our vision of PROLOG led us to isolate two main functions that directed our hardware vision.

Statistics of G. Berger Sabbatel [2] report that 40 % of the execution time is spent in the procedures relative to the environment management. 25 % of time is spent for resolution, 25 % of time is spent for unification.

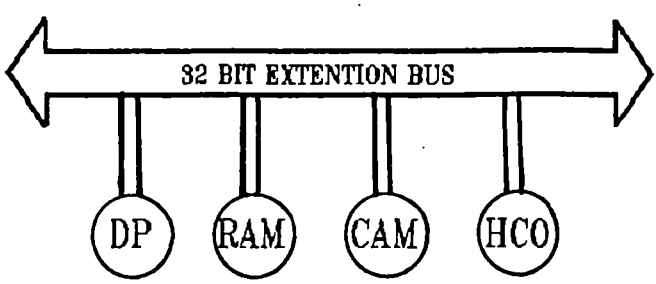
According to these, we found it was feasible to distinguish two types of processors at the hardware level.

2.2.1. Dispatching Processors (DP)

The designation came from the parallel view where DP distributes the tasks to other processors according to the load balancing.

According to the statistics above-mentionned, DP is in charge of the access to clauses, and realizes a preunification . He is connected to an Associative Memory Subsystem (AMS) for helping him in his clause access. AMS is made of a CAM and hash-coding operator detailed farther on (figure 2). The connection to the Transputer is done using a bus extension we designed. DP manages the hash tables and memory structures relative to the clauses.

Figure 2 : DP and its AMS

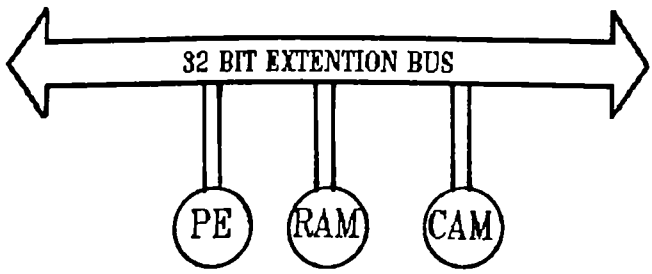


2.2.2. Processing elements (PE's)

On the same basis, we have PE realizing unification and management of the environments. PE is connected to a CAM with the bus extension we designed.

The CAM is used as a cache, to have a fast access to environments (figure 3).

Figure 3 : PE and its CAM



2.2.3. Sizing of hardware enhancements

The objectives were to determine the average number of clauses and arguments in PROLOG programs, in order to have usefull acceleration by hardware.

39 application programs were analysed by M. Py including 3736 clauses, 1501 facts and 2235 rules.

The mean value are 2.35 arguments in a clause, and 2.29 arguments in a packet. Furthermore 87 % of the packets have 3 or less arguments.

The interest of our research increases when big packet size are reached, i.e. in data base problems.

2.2.4. Hash Coding Operator (HCO)

In compiling PROLOG D.H.D. Warren realizes hashing on one argument. To be more efficient, our measures led us to have a hashing scheme with the 3 first arguments.

However for complexity problems we were led to realize that hashing only to access clauses whose 3 first arguments are constants. Other cases are managed by a filtering done by the CAM.

The HCO operates a polynomial division computation transforming a n-bits information into a m-bits results ($m < n$). The output result is used by the DP to address the memory part holding the hash-tables.

2.2.5. CAM

The CAM designed is general enough to be used in two different uses : preunification and variables access. They are cascadable to reach a total size of 16 K words of 32 bits. The design allows don't care fields for variables.

Preunification. The method is an extension of the one of Robinson [10] : our clause heads contain variables. CAM is well adapted to such clauses which are also far less numerous than those with constants in the case of knowledge base applications.

Variables access. We only store PROLOG variables that have been instanciated, not the whole environments. We create dependencies between variables and their environment. That allows a global reset when backtracking, and compared with traditional implementations avoids covering reference chains.

2.3. Hardware enhancements for LISP

Two enhancements have been studied for LISP : the CAM and a coprocessor.

2.3.1. CAM

The previously mentionned CAM can be profitably used to speed up LISP.

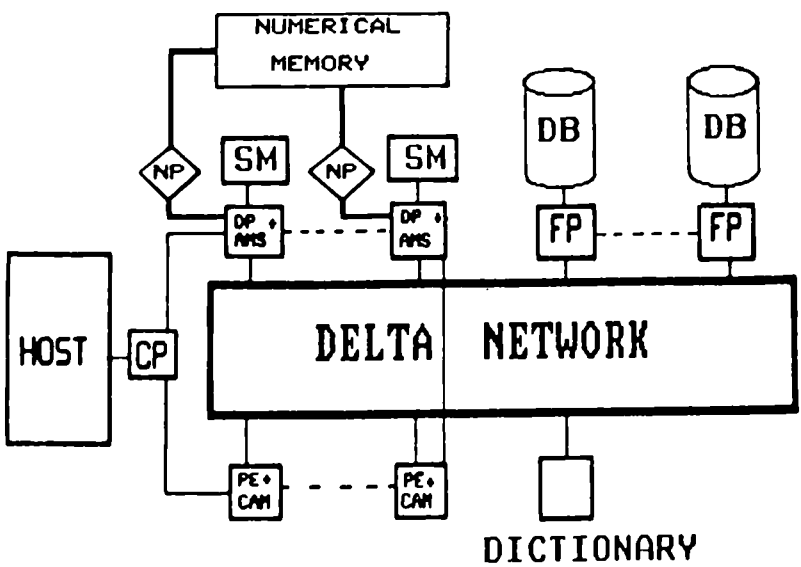
Two examples of functions that may benefit from the CAM are selector functions and memo functions.

2.3.2. LISP coprocessor

The main idea came from the observation that in many interpreters a lot of time is spent in a critical loop.

Only taking 5 to 10 % of the code, the execution of this loop enhanced the performance by an order of magnitude in a 68000 previous version.

Figure 4 : PADMAVATI global architecture including extensions



2.4. Knowledge based interface, symbolic and numeric integration

These studies were not to be implemented in the framework of this project. We adapted existing work to our context.

Concerning the knowledge base interface, we started from the work done in Grenoble [1]. PROLOG is seen as a support of relational algebra. We designed a filtering processor compatible with our parallel architecture and with the PROLOG model studied.

Symbolic and numeric integration gave way to a loose coupling solution. The work is continuing in another ESPRIT Project n° 1588 SPAN (Symbolic Processing And Numeric).

Figure 4 describes the complete architecture.

3. SOFTWARE ARCHITECTURE

3.1. Introduction

As pointed out in the introduction, software architecture was defined concerning parallel LISP and parallel PROLOG.

The LISP parallel model is based on a set of programs communicating results via messages.

In PROLOG, to start with we have investigated an analogous philosophy than LISP. Then we came to a more promising philosophy for PROLOG using Intelligent Backtracking [4]. It will be fully implemented in March 1989.
An intermediate stage, whose deadline is March 1988, is made of a PROLOG splitted on a pair of DP/PE.

In the parallel LISP or PROLOG models, the processes are synchronously communicating with run-time support processes that manage buffering and routing of messages. The MicroVax host supports the I/O and the file system. The debug is also managed by the host via VMS processes.
The OCCAM processes are simulated by VMS processes.

We will now describe the principal features of the computational models defined concurrently with the hardware.

3.2. LISP software architecture

Le_Lisp from INRIA [5] has been used as a starting point of our language. The computational model consists of a set of concurrent Lisp processes statically allocated into the processing elements.

The communication between these parallel processes is made through asynchronous SEND/RECEIVE primitives.

Le_Lisp LLM3 virtual machine is directly compiled into Transputer assembly code. This allows efficiency at each process level.

Each process is defined to have n input buffers, one for each message type. The sending process gives a priority to the messages to be queued.

The syntax of the non-blocking SEND primitive is :

```
(SEND proc-list buffer prior msg).
```

. proc-list : list of receiver process names.
. buffer : buffer name.
. prior : priority.
. msg : S - expression.

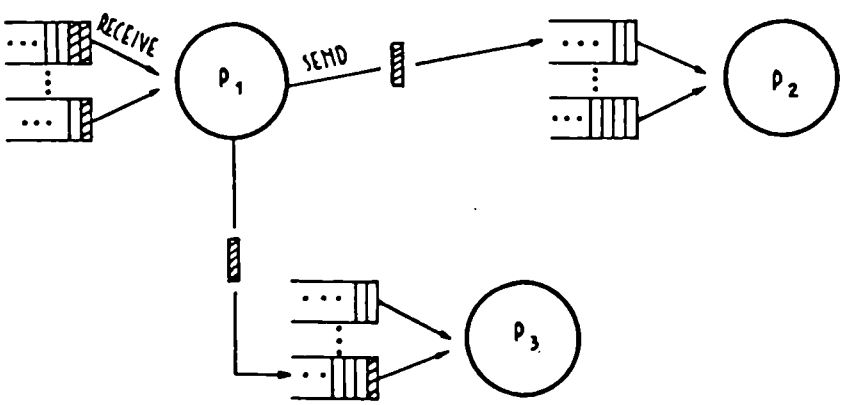
The syntax of the RECEIVE primitive is :

```
(RECEIVE var  (buffer - 1  body 1)
              (buffer - 2  body 2)

               [(body - else)]).
```

. var : name of the variable assigned to the received expression.
. buffer-i : name of the selected buffers.
. body-i : to be evaluated after receiving the corresponding message.
. body-else : form to be evaluated if no message is present in the selected buffer.

Figure 5 : LISP computational model



3.3. PROLOG software architecture

We will exibit to start with the ideas of the PROLOG splitted on a DP and a PE. The we will show the approaches studied and retained for parallelism.

3.3.1. Distributed PROLOG

This step had two goals :

- Simulation and validation of hash-coding and associative access independently of the parallelism.
- Evaluation of the gain obtained in such a splitting and the messages overhead.

Figure 6 summarizes the spitting.

This distribution allowed a rather straight forward repartition of the internal structures of the interpreter.

A linear representation of terms avoids conversions into an usual form to the serial links. Usually :

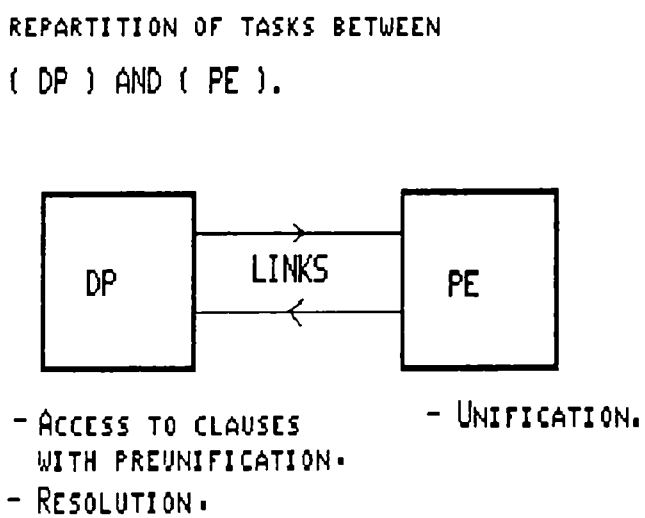
Transfer time = Linearization + Transmission + Delinearization

We have with the linear representation :

Transfer time = Transmission

Links allows transmission at a 20 M Bit/s rate.

Figure 6 : Splitting of PROLOG between DP and PE



This splitting allows preunification in advance. The chaining of clauses without variables (hash-coding tables) gives directly the address of the next preunified clause. For other clauses DP will be doing the filtering, while PE will be performing unification. That method allows suppression of choice points when not necessary.

Other easy speed up are obtained by :

. detection of the / in first position of the tail (no choice point to be created).

```
P (...) → /   q ;
P (...) → / ... ;
P (...) →   ... ;
```

. detection of empty tail (allows the next resolution to proceed immediatly).

3.3.2. Parallel PROLOG

We will report the different PROLOG models we have studied. The first paragraph reports a type of macro-parallelism. It has sharpen our views about distributed memory management but will not be implemented.

Then we will summarize the model defined for this project, based on distributed AND parallelism.

3.3.2.1. Initial parallel PROLOG studies

The simulations of this model were oriented in 3 steps. First, PROLOG was extended with OCCAM like communication primitives. No variables were shared between concurrent statically allocated processes. Then, distributed backtracking was introduced. To end with, we allowed dynamic creation of processes and chanels. That dynamic creation gave a power of expression greater than Delta Prolog [9].

However for semantic clarity, we shall not implement this version.

3.3.2.2 A parallel PROLOG model : LOGARITHM

We have kept the syntax and evaluable primitives of C-PROLOG. The "//" construct is added to indicate parallel evaluation of arguments.

Ex : P (...) // q (...).

Because of parallelism the use of the cut is not allowed in a subtree whose root is "//".

We rejected languages based on Guarded Horn Clauses, like PARLOG [6] or Concurrent Prolog [11]. The reason is we wanted to keep the non determinism and completeness of sequential PROLOG semantic.

We used Intelligent Backtracking [4] extended for parallelism. Using a dependency graph, it allows backtracking to the possible causes of failure, instead of taking the last choice point.

Computational model

The evaluation of a goal is managed by a process. Process are created for the subtrees of a AND parallel node.

Each process is made of a complete Warren abstract machine. Communication between processes are achieved by their common variables like in PARLOG [6].

Preliminary measures

According to simulations, in the map coloring problem, the extension speed

appears to be hundred times faster than the same problem written in PROLOG II.

Conclusion

LOGARITHM appears to be rather promising. However better performance ideas will be given in an implementation where clauses will be semi-compiled in Warren code extended to parallelism.

4. CONCLUSION

Let us summarize what has been done and what are the next stages hopes.

The hardware has been fixed and is starting to be built. The different computational models are implemented and critical parts are measured on the Transputer.

It is a major advantage of the Transputer to be able writing code on one processor, then splitting the code on several processors to obtain true parallelism.
Boards of 4 Transputers (B003-2) can be easily connected in a rack of 10 cards (B201-1).

The splitting of PROLOG on a DP and a PE under implementation will give us a good idea of the communication overhead, in case of a small grain of parallelism. Measures about PROLOG will also lead us to evaluate the best proportion between DP numbers and PE numbers. It will also help to refine our strategy of load distribution.

In the year 1989, studies of interactions with PROLOG III will show us how our architecture can accelerate what could be a major advance in logic programming. This year will also fully validate our developments with applications whose performances with more traditional vehicles are known.

5. ACKNOWLEGMENTS

I would like to thanks all the people that are related to the project in the EEC, in the various Companies and Universities, and that I cannot all mention.

REFERENCES

[1] G. BERGER SABBATEL, W. DANG, J.C. IANESELLI, G.T. NGUYEN, Unification for a Prolog database machine, (2nd Int. Logic Programming Conference, Uppsala - Jul. 84).

[2] G. BERGER SABBATEL, "Mesures comportementales sur l'interprêtation de PROLOG". Actes du séminaire TREGASTEL 86 (CNET LANNION 86).

[3] P.G. BOSCO, E. GIACHIN, G. GIANDONATO, G. MARTINENGO and C. RULLENT, A parallel architecture for signal understanding through inference on uncertain data, PARLE, volume 1 Parallel Architectures (Eindhoven, the Netherlands, June 15-19, 1987, Proceedings).

[4] M. BRUYNOOGHE, L.M. PEREIRA, "Deduction revision by Intelligent Backtracking", Implementations of PROLOG, (Ellis Harwood, 1984).

[5] J. CHAILLOUX, M. DEVIN, J. HULLOT, LeLisp a portable and efficient LISP system, Proc. of the 1984 ACM sym. on LISP and Functional Programming, (Austin Texas, August 1984), PP 113-122.

[6] K. CLARK and S. GREGORY, Parlog : A Parallel Logic Programming Language, (Imperial College London 1984, Research Report Doc 83/5).

[7] F. GIANNESINI, H. KANOUI, A. PASERO, M. VAN CANEGHEM, Prolog (Intereditions, Paris 1985).

[8] V. HERMENEGILDO, "Efficient management of backtracking in AND - Parallelism", (Lecture Notes in Computer Science, London, July 1986).

[9] L.M. PEREIRA, L. MONTEIRO, J. CUNHA, J. APARICIO, Delta PROLOG a distributed backtracking extension with events, University Nova de Lisboa, Third International Conference on Logic Programming 1986, (Spinger-Verlag).

[10] I. ROBINSON, A Prolog Processor based on a pattern matching memory device, Third International Conference on Logic Programming, (Lecture Notes in Computer Science, London, July 1986), PP 172-179.

[11] A. SHAFRIR, E. SHAPIRO, Distributed Programming in Concurrent PROLOG, (Tech. Rep. CS84 - 02 ICOT - 1984).

[12] THOMSON-CSF DIVISION CIMSA-SINTRA, CSELT, GEC, LER, NSL, PROLOGIA, UNIVERSITA DI TORINO, POLITECHNICO DI TORINO, (PADMAVATI Deliverable 1 on system definition, March 1987).

[13] M. VAN CANEGHEM, L'anatomie de PROLOG II, Thèse d'Etat, (Université d'Aix - Marseille 1984).

[14] I. VUONG, A. WOZNIAK, S. KRISHNA, I. FILOTTI, KOALA : A cost effective workstation for fast LISP interpretation, (Rapport de recherche LRI, Université Paris Sud).

[15] D. WARREN, "Implementing PROLOG", (Tech. report 39, Edinburg University, May 77).

[16] D.H.D. WARREN, "An abstract Prolog instruction set", (Technical note 309, SRI International, Menlo Park, California, October 1983).

Project No. 857

USER MODELLING IN THE GRADIENT PROJECT

Erik Hollnagel
Computer Resources International
Vesterbrogade 1A DK-1620 Copenhagen V, Denmark

George Weir
University of Strathclyde, Scottish HCI Centre
George House, 36 North Hanover Street, Glasgow G1 2AD, United Kingdom

Gunilla Sundström
Gesamthochschule Kassel, Fachbereich Maschinenbau
Labor. für Mensch-Maschine Systeme
Mönchebergstrasse 7 D-3500 Kassel, F. R. Germany

ABSTRACT

In the design of the GRADIENT system several user modelling strategies are required to provide flexible support not only across different operators but also to particular operators at various stages of expertise. Our approach is to separate the role for operator modelling according to tasks within the GRADIENT system. Thus, we distinguish between the need for modelling operator's plans at the level of control operations from the tasks of modelling the operator's strategies in dialogue. A third facet of modelling is addressed in designer support tools which complement the GRADIENT system. Here, the designer can employ an intelligent graphical editor which comes complete with insight on the target operators in terms of display ergonomics and domain dependent graphical standards.

1. THE GRADIENT PROJECT

GRADIENT (for GRAphical DIalogue environmENT) is a five year ESPRIT project to design and develop a graphics and knowledge based dialogue interface for industrial Supervision and Control (S&C) systems - process control, power distribution and data communication networks - and, more generally, monitoring of multiple, dynamic activities. The project has two main objectives. Firstly, to develop a prototype Graphical Dialogue Environment which can support operators in their monitoring and supervision tasks, and, secondly, to build a suite of software tools which support designers in implementing a particular GRADIENT application. In each of these aims, an essential requirement is the facility to model the end-user. In this paper we detail the role of such modelling in each of the GRADIENT objectives and argue for a distributed approach to modelling in a system of this nature.

1.1 General Background

Current S&C systems provide least support for the operator at those moments when it is most needed - during adverse conditions - and the dialogue between the system and the operator is usually inflexible and non-adaptive. If graphics display systems are used, they function by assembling pre-packaged pictures, to present information anticipated by the display designer.

In GRADIENT, we are investigating the use of knowledge-based systems to support operator decision making during normal and adverse system conditions. GRADIENT will use techniques from knowledge processing and user modelling to support a flexible, dynamic dialogue system. The operator of the S&C system will be given intelligent support during fault investigation, emergency cont inment, process tuning, and system modification. This is to be achieved by a set of interacting expert systems which will provide:

- o knowledge-based alarm analysis and fault identification,
- o recognition and monitoring of operator tasks and strategies,
- o intelligent control of the dialogue between the operator and the S&C system,
- o communication to the operator by dynamic generation of graphics displays.

The project is using a demonstrator S&C system to prototype, test, and evaluate each stage in the development of the knowledge-based modules. Construction of these component systems is being preceded by development of an improved S&C dialogue system and a set of intelligent design tools for use in building advanced graphics display systems.

2. SYSTEM ARCHITECTURE

The impetus for the Gradient project derives from two principal shortcomings of current control systems. Such systems provide little or no support to operators during emergency situations, and ordinarily maintain inflexible dialogue between operators and target systems (cf. Alty et al., 1985).

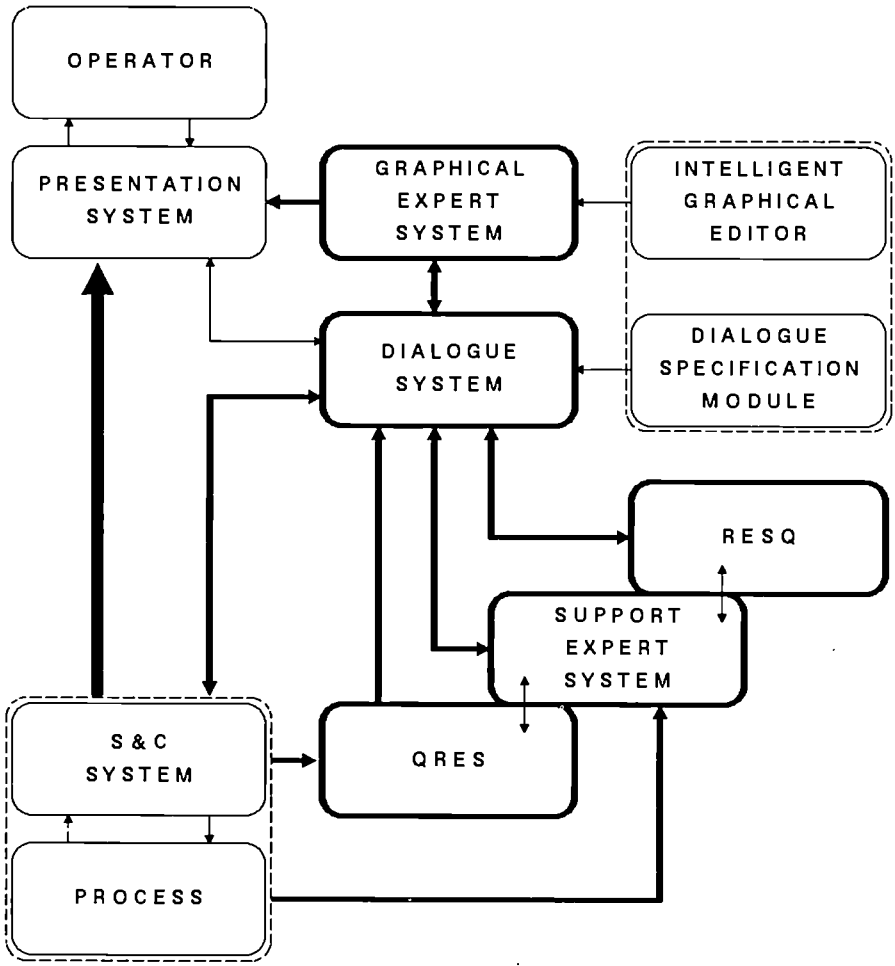
The Gradient design will consist of three co-operating expert systems which support an advanced dialogue system. Additionally, an intelligent graphical interface will enable the operator to handle a large volume of information from the control system. The graphical display will be formed and controlled by a graphical expert system, allowing it to decide not only what information should be displayed but also how it should be displayed intelligently. An overview of the system is given in Figure 1.

Flexibility in dialogue is achieved from the design phase through use of support tools for the dialogue and graphics designers. These tools maintain inbuilt knowledge relevant to the application domain, and are able to provide checks on the detail and consistency of dialogue and graphics specifications as they are composed.

Operator support is concentrated in three knowledge-based systems. The first of these, the Support Expert System (SES), acts as an intelligent consultant to both the operator and the other modules of the GRADIENT system. The remaining two act as intelligent monitors. The Quick Response Expert System (QRES) monitors key values from the process through the S&C system and handles alarms generated by the existing S&C system. Using a small, finely-tuned knowledge-base, QRES responds quickly to error syndromes, advises the operator of the most important condition and recommends possible restorative action.

The Response Evaluation System (RESQ) monitors operator actions in order to identify erroneous actions or potentially problematic trends in his behaviour. Any such indications will trigger warnings from RESQ to the operator.

At the heart of GRADIENT lies its Dialogue System. This system (DIS) is charged with control of communication between each of GRADIENT's advisory components and the operator(s). The DIS architecture must accommodate multi-threaded asynchronous interaction between operators and GRADIENT. In addition, it should be able to assist



ESPRIT #P857 – GRADIENT

General System Architecture

operators in framing queries to the SES, through its inbuilt knowledge of dialogue and of the operator's interaction with GRADIENT. (For more detail on the DIS architecture see Alty & Johannsen, 1987.)

3. USER MODELLING

In accord with the structure and functionality of the GRADIENT system, user modelling is relevant in several places. At the initial design phase of any GRADIENT implementation, the designer requires support with both dialogue and graphical specifications. In part, this takes the form of inbuilt operator models. In the Dialogue System, modelling is required in order to support operators with query formation and SES directed dialogue. Finally, RESQ must maintain models of operator action strategies in order to support its interpretation of operator behaviour. Details of the strategies employed for each of these requirements are given below.

3.1 User Modelling For Dialogue Designers

The requirements on the dialogue designer are quite specific. If the designer is to characterise adequately the possible range of interaction between operator and the GRADIENT system, he must have a master's grasp of the target process. Since the purpose of the dialogue system is to handle queries from operators and warnings and advice from the GRADIENT sub-systems, the individual who designs the dialogue must be acquainted with the range of possible queries that may arise in relation to the application domain. In turn, this knowledge allows him to specify operators' likely dialogue objectives, in reaction to what they may ask, and under which circumstances they may ask it. Effectively, the dialogue designer will have two tasks. Firstly, he must specify the range of possible queries and answers between operator and SES. This will be done using a high-level dialogue specification language as part of the dialogue design tool (cf. Alty & Mullin, 1987).

Subsequently, based upon his initial dialogue specification, the designer is required to characterise typical dialogue objectives or dialogue plans. These plans form the basis of a dialogue user model, in terms of which the GRADIENT Dialogue System (DIS) can lend support in operator interaction. In effect, this model will allow DIS to reason about the operator's likely concern, given the query options he selects in relation to specific process states. This overlay model of dialogue plans should serve not only the GRADIENT Dialogue System in its monitoring and operator prompting roles, but will also provide valuable insight for the graphical design module, where the graphics designer can be advised both of the sequence of dialogue and the likely dialogue contexts. Part of the role of the dialogue design tool is thus to provide the designer with support in formulating an operator model for dialogue. The dialogue design tool does not itself contain or employ any such model. In this respect it differs from its complementary graphics design support tool.

3.2 User Modelling For Graphics Designers

Current design of graphics for support of operators of dynamic systems is mainly based on the "common sense" of the person doing the design. In other words, the design process relies on a lay-man's hypotheses about the behaviour and the needs of operators. This situation can be improved by providing the designer with an explicit model pertaining to aspects of operators' problem solving behaviour in supervisory and control situations. The question arises as to which aspects of operators' behaviour may be modelled and which type of user model would be appropriate. A cognitive modelling approach, i.e. actually accounting for cognitive processes by simulation, should be based on an existing cognitive

theory. Since there is a deplorable lack of sufficiently developed cognitive theories, this is not a viable option at present.

Clearly, one source of insight into operator objectives is the dialogue design tool mentioned above. This will provide valuable information in the form of dialogue plans. This and other aspects of operator behaviour to be modelled must be related both to information processing behaviour and be of importance in existing theories of operators' problem solving behaviour. One feature which meets these two criteria is operators' information seeking strategies.

Information search has been stressed in all process tracing approaches (Newell & Simon, 1972) and also in the information processing approaches suggested by Rouse (1983) and Rasmussen (1984) within the field of man-machine studies. On this basis, it seems reasonable that the user model should provide the designer with general knowledge about operators' information search strategies as well as knowledge about factors that are known to influence information search. Obviously an operator model for use by the system designer should contain general knowledge about how operators process information in supervisory and control situations and not knowledge about any specific operator. This requires a canonical rather than an individual model (Rich, 1983).

In order to be able to structure knowledge about information search, behavioural descriptors are necessary. Operator behaviour may be described as either categorisation, planning, or action (cf. Rouse, 1983). In addition, the problem solving process is characterised as decision making where the decision alternatives are hypotheses about the state of the process (related to categorisation) or about possible actions (related to planning). The evaluation of the hypotheses gives rise to information search. Thus, the model provided to the designer is basically a choice model represented in a state-space (Simon, 1983). Nodes in the state space represent conditions for and consequences of actions. In order to be able to represent the choice model in the state-space, knowledge about the objects and their relations within this state-space has to be accumulated. This is derived by task analyses in the particular application domain for any GRADIENT implementation.

Such explicit knowledge on operator behaviour can be used by the designer in the course of designing the pictures and display sequences which will eventually be presented to the process operators. The canonical user model will act as a pre-programmed image of a typical operator and act as a standard against which to assess and advise the designer of the graphical specification.

3.3 User Modelling In Dialogue

In GRADIENT, the Dialogue System is concerned to follow the interaction of operators with SES and in their selection of graphical information screens, in order to offer the most appropriate or most efficient dialogue options for the given context. At the same time, DIS attempts to make operator interaction as flexible as possible. The latter is achieved through the use of Dialogue Assistants, which constitute packets of dialogue or conversations on specific aspects of SES interaction. Access to these assistants is structured such that various levels, including a top-level assistant, are always available to the operator, who can thereby pursue any line of enquiry at will. In addition, interaction with SES on any topic may be discontinued whenever the operator chooses. (See Alty & Mullin (1987), for more on Dialogue Assistants.)

Dialogue options are made available to operators by menu selection. While it will always be possible for the operator to invoke a desired option, the precise combination of menu

options presented at any one time may be varied in accord with insight from the dialogue user model. Thus, if the operator's interaction with SES indicates a likely interest in a specific process sub-system, the menu option to view this sub-system in detail will be

assigned greater priority. Thereby, the user model provides advice to the presentation system on which dialogue routes immediately to offer the operator in his exchanges with SES.

3.4 User Modelling In Action Monitoring

The principal role for RESQ within the GRADIENT architecture, is in the monitoring and analysis of operator actions. RESQ performs this function in terms of operator plans. A plan is defined as consisting of a goal and an activity which is one or more actions in an ordered sequence. For any given situation the operator is assumed to have a goal or target. A task is defined as the activity (or sequence of actions), described on the level of generalised functions, by which the man- machine system can reach the goal. If the goal is simple, a single task may suffice. But if the goal is complex, and possibly composed of separate sub-goals, a set of tasks or a more complex activity may be required.

The individual actions in an activity may be either unique or replaceable, and one or more steps in an activity may be either required or optional. This makes it rather difficult to produce a generic description of an activity, let alone trying to follow it. However, the situation is further complicated because individual actions may match more than one activity (one-to-many mapping), just as more than one action may match a step in a activity (many-to-one mapping).

The general method used in RESQ is based on a description of the set of potential plans (goal-activity lists). These must be provided in advance by the system designer, either based on the design specification or on extensive knowledge elicitation. Within each goal-activity list the individual actions must be marked as being replaceable/unique or required/optional. (This may, of course, also be the case for sub-sequences consisting of several actions, but these may then be regarded as a single conglomerate action in relation to the particular goal). Initially, only one or a few goals are active. At least one must be active, but depending on how the situation is initiated there may also be two or more. The active plan is transferred to a separate knowledge base. Here it will be interrogated whenever an event takes place, i.e. whenever there is input to the system.

In the simplest possible case, a new action is checked against the next expected action in one of the current active plans. If the action matches the step it is marked in the plan, which is further checked for completion. If the action does not match, the set of passive plans are searched to see if the action fits into any of them as the first step, in which case that particular plan is added to the set of active pans. This must, of course, be done with due regard to whether an action is unique/replaceable or required/optional. This alone makes the scheme far from straightforward to implement.

Clearly, there are limitations to this simplistic approach. What happens, for instance, if the action does not match any known plan? It could be because the set of plans is incomplete; because the action is misplaced, e.g. a human error as in a slip; or because the action is simply wrong, i.e. a genuine error or mistake. Furthermore, the position of an action in a task sequence may not be unique in isolation, but only when seen in relation to the actions that precede and follow it, i.e. it may be necessary to keep several possible orderings 'alive' at any one time. A complete solution must obviously be based on a theory or taxonomy of human error and make heavy use of the reasoning techniques developed in artificial intelligence applications, particularly the idea of 'possible worlds'.

4. DISCUSSION

A notable feature of our approach to operator modelling is the distribution of models in accord with the modelling objectives of particular GRADIENT modules. Thus, we have one model in the Design Tools, one in the Dialogue System and a third in RESQ. Why have three models when, ostensibly, one would do?

In the first place, the requirement for user modelling at the design stage, when both the dialogue and graphics specifications are defined, involves us in deriving part of the user model from the dialogue and graphics designer. This source of operator insight, supplemented by information from knowledge acquisition and task analysis and also an account of operator's information search strategies, is employed off-line from the particular GRADIENT application. In contrast to the two on-line models, the user model available for the design of the graphics does not change as a function of who is using the system. In consequence, focus is less on the detection of current operator's objectives than on representing key factors of operator behaviour as well as establishing a relation to the process of designing graphics.

The requirements for the on-line user modelling are very different. In the latter group of models, the concept of plans (behavioural or dialogue), as well as their detection, are of central importance. Thus both the user model in DIS and in RESQ focus on monitoring current behaviour of operators. In RESQ, the need for user modelling centres on monitoring operator actions in real time. As such, while dialogue plans may be of some value, in terms of insight into operator's purpose, they will be substituted by behavioural plans that account for the dominant patterns of interaction with the process. The main difference between the user models in RESQ and DIS is that the latter centers on dialogue objectives and dialogue plans, while the former focusses on behavioural plans with the intention of correcting operator behaviour. The user models envisaged within GRADIENT differ in the purposes they serve. In itself, this gives good reason for maintaining distinct models in such a complex interactive system. Reflecting their different objectives, the user models in RESQ and DIS will employ different knowledge bases. RESQ is concerned with the user's behaviour in relation to control actions, whereas DIS attends to the operator's dialogue with the GRADIENT advisory systems. Clearly, these require different insights. In turn, this difference in knowledge requirements will be reflected in the use of different inference mechanisms.

While there may be a connection between an operator's objectives in dialogue with GRADIENT and his control objectives for the S&C system, the complexity, not to say, opacity of such relations set them outside the scope of current modelling techniques. To some degree, this reflects the lack of a generally recognised theory of cognitive user modelling. In the absence of such an approach, functional separation of user models is the only viable alternative.

5. ACKNOWLEDGEMENTS

The work reported here was carried out under ESPRIT Project #857 in a consortium consisting of CRI (Copenhagen, Denmark), BBC (Heidelberg, F. R. Germany), University of Kassel (Kassel, F. R. Germany), University of Strathclyde (Glasgow, Scotland), and University of Leuven (Leuven, Belgium). The contributions of all members of the project consortium are gratefully acknowledged.

6. REFERENCES

Alty, J. L., Elzer, P., Holst, O., Johannsen, G. & Savory, S. (1985). *Literature and user survey of issues related to man-machine interfaces for supervision and control systems.* Copenhagen, Denmark: Computer Resources International (ESPRIT P600 Final Report).

Alty, J. L., & Johannsen, G. (1987). *Knowledge based dialogue for dynamic systems.* Survey paper IFAC 10th World Congress, Munchen, F. R. Germany.

Alty, J. L. & Mullin, J., (1987), The Role of the Dialogue System in a User Interface Management System. *Proceedings of Interact' 87.*

Newell, A. & Simon, H. A. (1972). *Human Problem Solving.* Englewood Cliffs, N.J.: Prentice Hall.

Rasmussen, J. (1984). Strategies for state identification and diagnosis in supervisory control tasks, and design of computer based support systems. In, W.B. Rouse (Ed.), *Advances in man-machine systems research.* Vol. 1. Greenwich: JAI Press.

Rich, E. (1983). Users are individuals: individualizing user models. *International Journal of Man-Machine Studies, 18,* 199-214.

Rouse, W. B. (1983). Models of human problem solving: Detection, diagnosis, and compensation for system failures. *Automatica, 19,* 613-625.

Simon, H. A. (1983). Search and reasoning in problem solving. *Artificial Intelligence, 21,* 7-29.

Project No. 857

DIALOGUE DESIGN FOR DYNAMIC SYSTEMS

J L Alty and G R S Weir

Scottish HCI Centre, University of Strathclyde, George House, 36 North Hanover Street, Glasgow G1 2AD, Great Britain

The importance of separating dialogue for application in an interactive system is emphasised. The Dialogue Controller architecture proposed for the ESPRIT GRADIENT system is outlined. This provides an intelligent interface to dynamic and supervisory systems. The approach is based upon a multi-channel dialogue handler, the dialogue knowledge being separated out in a series of knowledge-based dialogue assistants. The relationship between the dialogue controller and the other modules in the GRADIENT architecture is discussed.

1. INTRODUCTION

A major effort in the GRADIENT project (P857) is directed toward improvements in the design of dialogue for complex dynamic systems. The need for improvements in this area became evident in an earlier pilot phase project (P600). In the pilot phase, an operator and literature survey indicated that current dialogue systems for process control were lacking both in adaptability and flexibility (cf. Alty, Elzer, Holst, Johannsen and Savory, 1985). These problems were undoubtedly due in part to the complexity of design for such large-scale applications.

In the face of such complex design requirements, one line of attack from our project is the development of designer support tools for dialogue and graphical specification. The main weapon for assault on the flexibility problem is the architecture of the GRADIENT dialogue system (DIS) itself. Both of these approaches depend upon the introduction of knowledge based techniques. In elucidation of our strategy, the present paper, details the benefits arising from functional separation of dialogue from tasks and the design requirements placed upon the DIS architecture by the exigencies of dynamic systems.

2. DIALOGUE SEPARATION

The governing principle underlying dialogue specification is that dialogue should be definable separately from the tasks in the application. Although this seems desirable, interactive dialogue has in general, been regarded simply as part of the programming code of the application, initiated from input/output statements embedded in conventional languages. This has a major drawback if one wants to vary the way in which user and application communicate. Whilst variability in the communication can be achieved through extensive use of conditionals the approach is clumsy. It is not easy, for example, to record user interactions for future analysis, the dialogue cannot adapt to different user circumstances and any changes to the dialogue result in major programming updates. Furthermore no analysis can be performed on the dialogue. Edmonds (1982) suggested that the dialogue aspects of an interaction should be separable from the application code, and early dialogue systems such as CONNECT (Alty and Brooks, 1986), SYNICS (Edmonds and Guest, 1984) and RAPID

(Wasserman and Shaw, 1983) provided dialogue creation facilities which supported this separation. Generally, a "dialogue controller" is driven by a specification of the dialogue and this controller communicates both with the user and with the application. Techniques used for specifying the dialogue included transition nets (Woods, 1970), production rules (Waterman, 1978), event-driven approaches (Green, 1986) and Command and Control Languages such as TICCL (Kasik, 1982).

The benefits from functional separation of dialogue and application code are considerable. For example, the set of actions can be logged for play-back or for analysis. This is important given the current pragmatic nature of dialogue design. The dialogue can be readily changed without altering the application code. Different dialogue sequences can be provided to cope with users of differing experience and a limited form of on-line adaptability can be supported (see the CONNECT system for example (Alty and Brooks, 1986)).

Importantly, such functional separation allows the dialogue to be cast in the form of a specification language which can in turn be analysed (for example using Path Algebras (Alty and Ritchie, 1986)) and used for prototyping. Specification techniques include transition network specifications, modified forms of BNF notation (Reisner 1981) and multiparty grammars (Shneiderman 1982). See Jacob (1983), for a review of dialogue specification techniques. For a critical evaluation of dialogue specification techniques and their possible use in adaptable systems see Cockton (1987).

This paradigm of dialogue-task separation enables support tools to be provided which make the construction of the dialogue a much easier task. Examples include interactive net construction in CONNECT, graphical construction of dialogues as in RAPID and a complete interactive workstation approach as in the FLAIR system (Wong and Reid, 1982). Yet this approach still suffers from a number of weaknesses. The dialogue definition is usually task dependent. In other words the sequence of actions on the net (or set of rules) have an intimate dependency with the task being performed. Furthermore, any implemented adaptability (possibly supported by some form of user modelling) is also implicit in the specification of the dialogue. Finally, presentation issues are normally mixed in with the dialogue specification as well. Thus this approach solves some problems but leaves others untouched.

In order to gain further benefit from the dialogue-task separation the different functions of the interface may be more clearly separated. One strategy is the User Interface Management System (UIMS) architecture (cf. Edmonds (1982), Green (1985), Pfaff & ten Hagen (1985)). The UIMS approach distinguishes the following functional components of a user interface system.

The Presentation System - This is responsible for the external presentation of the user interface i.e. how a particular request or interaction will be carried out in the hardware available. It does not concern itself with which particular presentation technique has been chosen. Naturally, this module is heavily dependent upon the actual hardware used in the interaction.

The User Model - This module advises the Dialogue System on the interaction style which should be used to communicate with a particular user or category of user, based upon its knowledge of the user in question. It might also act as an interpreter between the plans of the application and those of the user. Plans in the User Model would, of course eventually map onto application plans.

The Dialogue System - This system controls all interaction between the other modules. It serves as a form of telephone exchange and also holds general dialogue knowledge about the objects being manipulated and presentation styles available. It will normally have some application-dependent information.

The Application Model - This takes the form of an explicit representation of the tasks in the application and thereby allows the Dialogue System to reason about aspects of the application.

In the design of the GRADIENT system, we have adopted these principles of functional separation at several levels. In the first place, our Dialogue Design Tool will employ a top-down dialogue specification technique. This secures the benefits of dialogue-task separation detailed above. Furthermore, this tool will support the designer as he specifies the dialogue, by providing logical checks and analysis on his dialogue specification. Assistance will also be available towards integrating dialogue and graphical specifications for a single application.

Our approach to dialogue design, based upon functional separation, is further reflected in the overall design of the GRADIENT system. In keeping with the UIMS architecture, the GRADIENT design has separate modules for presentation, dialogue and application model. This ensures the greatest possibility of flexible control over display, dialogue content, and mode of addressing the application. Flexibility in dialogue is secured jointly by this separation and by the design of our Dialogue System.

3. DIS ARCHITECTURE

Process control imposes additional constraints on dialogue design compared with traditional human-computer interaction. It requires a rapid response time in critical situations, and has a large number of continuously changing state variables. Additionally, the Supervision and Control (S&C) System is fronted by a team of operators working on multiple Visual Display Units. In the full GRADIENT system there must be interaction between the operators, the dynamic system, its automation (S&C system) and all the intelligent interface and operator support modules. In order to simplify handling all such interactions a further functional separation is made between low-level dialogues processed by the Presentation System and high-level dialogues through the Dialogue system. The former low-level interaction involves direct display of process variables and handles operators' manipulation of the process. Although the latter is monitored by the Dialogue System, it is infeasible to manage process values, with their rapid variation, through the main Dialogue System. Likewise, in order not to delay operator control over the process, a direct link with the S&C system will always be available to the operator.

The central role for the Dialogue System lies in handling interaction between operators and the GRADIENT advisory systems. This requires that numerous conversation channels be available to the several operators. Furthermore, these channels must operate concurrently, interfacing each operator to the Support Expert System, which is part of GRADIENT, or interfacing GRADIENT's operator warning systems to output channels.
With this in mind, the DIS architecture has to meet three principal objectives. It should provide interruptable handling of conversation channels and should support the high-level dialogue specification technique employed in the Dialogue Design Tool. Thirdly, it must provide flexibility in dialogue such that operators no longer feel constrained or trapped in interaction sequences (cf. Alty, et al, 1985).

The first of these features is secured by including an Interrupt Handler and Scheduler in the Dialogue system. Additionally, a Dialogue event Handler controls a particular interaction once initiated. Different 'conversations' are made possible by use of 'Dialogue Assistants'. Dialogue Assistents contain the knowledge necessary to conduct interaction between operator and SES, or between a GRADIENT alarm system and an operator. In brief, they are used by the Dialogue System to handle application dependent conversations. Assistants will form a hierarchy with some assistants acting on the knowledge of others. Just as the operator can interrupt when he wishes a specific service, to call a particular assistant, Dialogue Assistants are able to interrupt the Dialogue System when further services are required, i.e. from other assistants. This approach has been termed the 'event model' (Green, 1985), and provides advantages such as allowing operators to conduct parallel dialogues with different assistants. This asynchronous approach to the DIS architecture is illustrated in Figure 1, below.

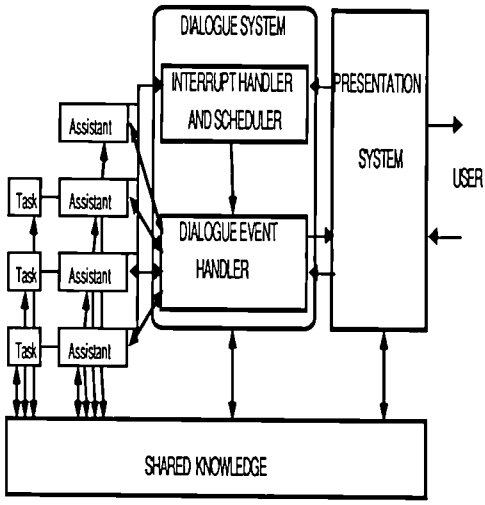


FIGURE 1 · INTERRUPTABLE DIALOGUE SYSTEM

When the system is initiated, one Dialogue Assistant will act as a "top level assistant". This will present the first level of user options to the operator, and enable the dialogue to commence. Thereafter, different assistants will come into play as required. This assistant concept is a general one and there will be assistants to help with presentation issues as well as task execution.

Currently, Dialogue Assistents are being implemented as event-driven networks in order to capture both the event and sequence nature of interactions. We have used a transition network interpreter/compiler in Franz Lisp (Ritchie, 1987) which overcomes a major limitation of other systems such as RAPID. This is running on a Texas Instruments Explorer Lisp-machine in KEE3 (software by Intellicorp). By regarding networks, nodes, local and parameter variables as objects they can be reasoned about in the high-level KEE environment.

Clearly, Dialogue Assistants are the vehicle for our dialogue specification. Their contents, which define individual conversations, are determined by the dialogue designer, and implemented via our Dialogue Design Tool. Dialogue flexibility is assured by means of the 'dialogue packaging' offered by Dialogue Assistants. The limitations inevitably imposed by sequence are never such as to restrict operators from pursuing other dialogues with the system. In every case, an active dialogue can be terminated or suspended by the operator in favour of some other interaction.

Figure 2, below, illustrates the structure of the Dialogue System within the GRADIENT architecture. In this representation, the Application Model comprises the Support Expert System, which can reason about the Process, and two operator alarm systems QRES and RESQ. QRES monitors the Process and advises the operator when urgent alarm conditions arise. RESQ monitors operator input and advises on worrying trends or possible pitfalls in such interaction. Both QRES and RESQ have dedicated Dialogue Assistants which respond to their output, and interrupt the Dialogue System accordingly. A Channel Sequencer Module is required to control access of Dialogue Assistants to the SES, since many assistants may be in conversation 'at one time'. The Knowledge Pool contains knowledge shared between several of the GRADIENT modules, including the Dialogue system.

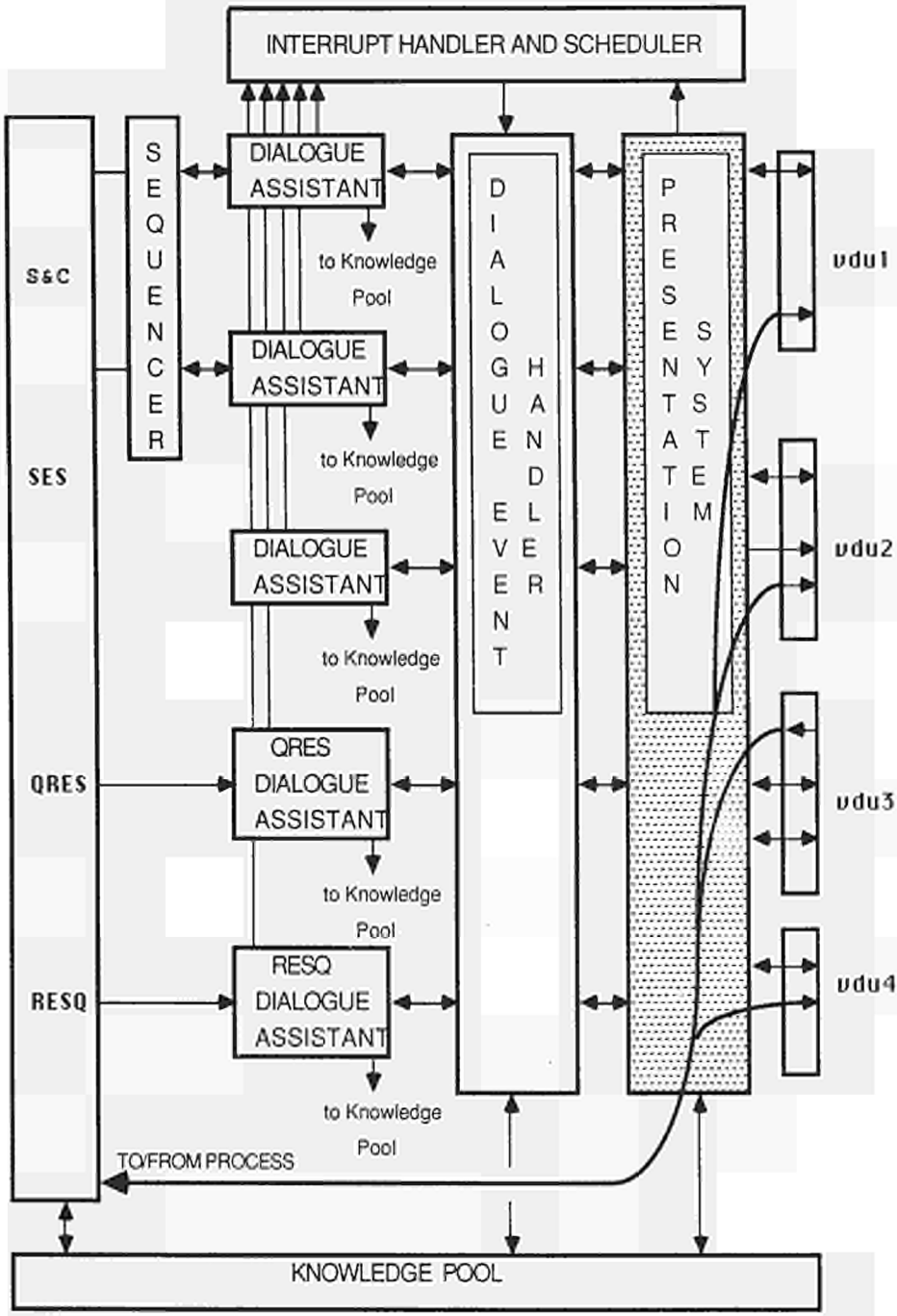


FIGURE 2 - THE GRADIENT DIALOGUE SYSTEM

REFERENCES

Alty, J.L., (1984), The application of Path Algebras to interactive dialogue design, Behaviour and Information Tech., Vol 3, No 2, pp 119 - 132.

Alty, J.L., and Brooks, A., (1985), Microtechnology and user friendly systems: the CONNECT dialogue executor, J. Microcomputer Applic., Vol 8, pp 333 -ɔ346.

Alty, J.L., Elzer, P., Holst, O., Johannsen, G., and Savory, S., (1985), "Literature and User Survey of Issues Related to Man-Machine Interfaces for Supervision and Control Systems", ESPRIT P600, Pilot Phase Report. (Issued as Scottish HCI Centre Report No. AMU8603/01S, 1986.)

Alty, J.L., and Johannsen, G., (1987), Knowledge Based Dialogue for Dynamic Systems, to appear in Proc. of the 10th. IFAC World Congress on Man-Machine Systems (Munich, 1987).

Alty, J.L., and Mullin, J. (1987), The Role of the Dialogue System in a User Interface Management System, in Proc. of Interact'87.

Alty, J.L., and Ritchie, R, (1986), A path algebra support facility for interactive dialogue designers, In People and Computers (ed. Johnson, P and Cook, S.,),Univ. of Cambridge Press, pp 128 - 137.

Cockton, G., (1987), Abstractions for Adaptable Dialogue Specification, in Proc. HCI'87.

Edmonds, E.A., (1982), The Man-Computer Interface: a note on concepts and design, Int J. Man-Mach. Studies, Vol 16, pp 231 - 236.

Edmonds E.A., and Guest, S.P., (1984), The SYNICS2 user interface manager, INTERACT '84, First IFIP Conference on Human-Computer Interaction, Vol 1, pp 53 - 56.

Green, M., (1985), The University of Alberta User Interface management system, Proc. SIGGRAPH '85, Vol 19, No 3, pp 205 - 213.

Jacob, R.J.K., (1983), Survey and examples of specification techniques for user-computer interfaces, Naval Research Laboratory, Washington, D.C.

Kasik, D.J., (1982), A User Interface Management System, Computer Graphics, Vol 16, No 3, pp 99 - 106.

Pfaff G., and ten Hagen P.J.W., (1985), Seeheim Workshop on User interface Management Systems, Springer-Verlag, Berlin.

Reisner, P., (1981), Formal grammar and human factors design of an interactive graphics system, IEEE Trans. Soft. Eng. SE-7, pp 229 - 240.

Ritchie, R., (1987), Private Communication.

Shneiderman B., (1982), Multiparty Grammars and Related Features for Defining Interactive Systems, IEEE Trans on Systems, Man and Cybernetics, Vol 12, No 2, March-April 1982, pp 148 - 154.

Wasserman, A.I., and Shaw, D.T., (1983), A Rapid/Use Tutorial , Medical Info. Science, U.C., San Fransisco, Calif.

Waterman, D.A., (1978), A rule based approach to knowledge acquisition for man machine interface programmes, Int. J. Man-Mach. Studies, Vol 10, pp 693 - 711.

Wong P.S.C., and Reid E.R., (1982), FLAIR - User Interface Dialogue Design Tool, Computer Graphics, Vol 16, No 3, pp 87 - 98.

Woods, W.A., (1970), Transition Network Grammars for Natural language Analysis, Comm. ACM, Vol 13, No. 10, pp 591 - 606.

Project No. 280

TEXT GENERATION IN THE EUROHELP PROJECT : THE UTTERANCE GENERATOR

Lene Stausholm

CRI A/S, Copenhagen
Vesterbrogade 1A
DK-1620 Copenhagen V, Denmark*

The user interface to an Intelligent Help System has a requirement to provide help in the form of natural language help texts. In the EUROHELP project, the responsibility for producing the natural language form of the help is delegated to a component called the Utterance Generator. This paper describes the EUROHELP framework for text generation and the current implementation of a prototype of the Utterance Generator.

1. INTRODUCTION.

1.1. The EUROHELP Project.

The aim of the EUROHELP project is to establish a methodology for providing intelligent help facilities for users of information processing systems and to operationalize this methodology by developing a Help System Development System which provides a tool for easy design and implementation of Intelligent Help Systems (IHS).

The IHS for a given Target Application (TA) must be able to interpret the user's performance in order to interrupt when something goes wrong or when there is an opportunity to extend the user's knowledge of the TA. Furthermore, the IHS must be able to answer questions from the user taking account of the context of his current interaction with the TA.

Thus, the help facilities of the IHS are both active and passive - active, when the IHS offers the user advice, passive, when the user initiates a help session by asking the IHS a question.

1.2. Requirements of the Help provided by the IHS.

The help presented to the user must fulfill a number of requirements, of content, form, and presentation in order to be easily understood and thus fulfill the goal of the IHS: to help and teach the user of a specific TA.

---

* Partners of the EUROHELP project (Esprit P280) are: CRI A/S (Denmark), DDC (Denmark), Courseware Europe (Netherlands), University of Amsterdam (Netherlands), University of Leeds (UK), and ICL (UK).

Concerning the form of the help, the IHS must be able to present it in the form of Natural Language (NL) help texts, as NL is one of the most effective means of communication. Whenever appropriate, help in form of graphics should be used; this has however not been investigated within the project so far.

The most basic requirements of help in the form of NL help texts are listed in the following:

Content:
- the information presented to the user must be relevant, i.e. take account of the context of his current interaction with the TA
- the help text must be based on concepts the user understands

Formulation:
- help texts must be syntactically correct
- help texts must be linguistically coherent

Presentation:
- the presentation of the help text must be clear and agreeable

1.3. Text Generation in the EUROHELP Project.

The EUROHELP project is well-suited for text generation:

First, the NL used in help texts represents a sub-set of "real" NL, i.e. the vocabulary is restricted and the syntactic structures are simpler than in "real" NL.

Furthermore, one of the (many) difficult issues of text generation, deciding **what** to say, which requires a knowledge base containing the relevant information and heuristics for retrieving it, is taken care of, as there is a representation of the individual TA, the Application Model, which other components of the IHS use for deciding the content of the help texts.

Thus, in the EUROHELP project, the requirements to the content and formulation of help texts are fulfilled by generating help texts using :

- a representation of the TA, the Application Model, common for all the components of the IHS
- knowledge about the user contained in the User Model
- knowledge about linguistics contained in the Utterance Generator

Compared to other approaches to production of help texts, such as on-line manuals (canned text) or slot-filling in text-frames without linguistic control, the EUROHELP-approach provides a number of advantages:

First, it ensures the relevance of the information contained in the help texts and the linguistic quality of the help texts.

Second, keeping in mind that the aim of the EUROHELP project is not to develop just **one** IHS but a Help System Development System for generating **several** IHSs, it provides a flexible way of producing the help texts.

Compared to on-line manuals, for which all help texts must be typed in, the IHS-developer (i.e. the person using the Help System Development System to generate an IHS) will have to do a lot less typing.

The linguistic rules are not dependent on a specific TA, i.e. they are general, and may therefore be used to generate help texts for different TAs.

During the generation process, text templates representing sentence patterns are filled in with NL words. This also provides flexibility since the sentence patterns and the NL words in the dictionary are re-used for different help texts.

Finally, the Application Model and the linguistic rules may be used for generating different kinds of texts.

In the EUROHELP project, the responsibility for generating the NL help texts is delegated to a component called the Utterance Generator. The Utterance Generator is only concerned with the linguistic realization of the help texts. The reason(s) for the user's need for help and the content of the help texts are determined by other components of the IHS. The rest of this paper is focused on the current implementation of a prototype of the Utterance Generator.

2. THE PROTOTYPE OF THE UTTERANCE GENERATOR.

2.1. Background Information.

The purpose of implementing a prototype of the Utterance Generator was to evaluate the structure of the Utterance Generator and the linguistic rules controlling the generation process described in Stausholm [1].

As the focus of the prototype was the linguistic aspects of text generation, the prototype was implemented as a stand-alone piece of software with no connection to any other components of the IHS.

The prototype was developed in the fast prototyping environment offered by a XEROX 1186 with InterLisp-D/LOOPS.

The prototype generates NL help texts in English for a sub-part of Unix Mail. The structure of the Utterance Generator is NL independent. Therefore, presupposing that language dependent parts such as NL words in the dictionary and some of the linguistic rules controlling the generation process are modified, the Utterance Generator may be used to generate help texts in other NLs.

2.2. The Architecture of the Prototype.

As the prototype was not connected to the components of the IHS determining the content of the help text nor to an Application Model for a TA, it not only had to contain the linguistic knowledge belonging to the Utterance Generator proper but also an Application Model (in this case for a sub-part of Unix Mail) and an interactive interface allowing the user of the prototype to decide the content of the help text.

Figure 1 below illustrates the architecture of the prototype (the arrows indicate the flow of information):

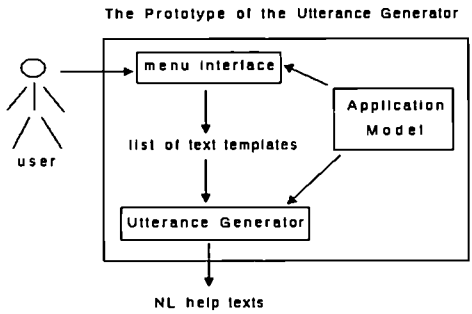


Figure 1

2.3. The Application Model.

The Application Model includes all the information specific to a particular TA. Since a full-scale Application Model would be time-consuming to develop without tools to facilitate this, and issues related to the Application Model had low priority in the implementation of the prototype, the Application Model developed for the prototype only describes a sub-part of Unix Mail.

The Application Model may be regarded as a network representation of Unix Mail, describing concepts (their attributes, if commands their parameters and effect etc.) and relations between them.

Figure 2 below** shows the linking of the concepts "UserName" and "Char" via the relation "consists_of". The relation is furthermore described via the cardinality (1,8) which means that a username (in Unix Mail) may consist of one to eight characters.

** The characters "#$" occurring in the examples throughout this paper are LOOPS-characters which indicate that the specific concept is an instance of an object.

```
#$Consists-ofUserName:
((First #$UserName)
 (Last  #$Char)
 (Min   1)
 (Max   8))
```

Figure 2

In addition to this kind of information, the concepts described in the Application Model contain a reference to the NL word in the Dictionary representing the translation of the term used in the Application Model into NL.

2.4. The Interface.

The interactive interface, which is designed as a menu-interface, allows the user of the prototype to act as the components of the IHS which determine the content of the help texts.

The concepts and their relations from the Application Model are presented to the user in menus. The user's indication of which concepts and relations he wants to serve as content of the help text triggers the generation of the input to the Utterance Generator.

For each selected relation a text template representing an appropriate sentence pattern is chosen, for example:

```
is_a --> ((Subject) (Verb) (SubjectComplement))
```

Figure 3

The concepts selected from the Application Model and a reference to the appropriate NL verb in the dictionary are added to the sentence pattern. Furthermore, the chosen relation is added to the sentence pattern as its first element. This allows the Utterance Generator to retrieve information about the chosen relation from the Application Model during the generation process.

As concepts and relations are chosen and templates filled in, a list of filled-in templates is generated. This list serves as input to the Utterance Generator. For example, the selection of the concept "UserName" and the relations "is_a" and "consists_of" gives the following input to the Utterance Generator:

```
((#$Is_aUserName
      (Subject #$UserName) (Verb #$BeVerb) (SubjectComp #$String))
 (#$Consists_ofUserName
      (Subject #$UserName) (Verb #$ConsistVerb) (Object #$Char)))
```

Figure 4

3. THE UTTERANCE GENERATOR PROPER.

3.1. Architecture of the Utterance Generator.

The Utterance Generator is composed out of a Dictionary, Interpretation Rules, Linguistic Rules, and Linearization Rules. This architecture is inspired by Rubinoff [2].

The **Dictionary** contains NL words. The **Interpretation Rules** are used to interpret the information in the Application Model. The **Linguistic Rules** consist of **Cohesion Rules** controlling the application of different means of connecting sentences and **Syntax Rules**. The **Linearization Rules** control the generation of the final form of the help text (i.e. punctuation, lower/upper-case letters etc.). The latter are not described in detail in this paper.

3.2. The Dictionary.

The Dictionary stores the NL words. As the English morphology is relatively simple, i.e. the words have very few forms, the Dictionary stores all the different forms of words needed.

The format of the dictionary entries vary according to the word-classes. The following figure shows the format of noun entries:

```
#$DirectoryNoun:
((Singular "DIRECTORY")
 (Plural   "DIRECTORIES")
 (IndefArt "A"))
```

Figure 5

The noun entries contain the indefinite article in order to ensure use of the correct morphological form of the indefinite article which would otherwise require morphological analysis of the specific noun. For example, the expression "Man Machine Interface" requires the form "a" whereas the acronym "MMI" also beginning with an "M" requires "an".

Verb entries contain present tense, singular and plural, and past tense. The verb "be" is defined as a specific word-class as it - in contrast to other English verbs - has both a singular and a plural form in the past tense.

The personal pronouns also represent an example of a NL word class which has been further specialized in this implementation, since some of them have specific objective forms (e.g. they - them) and others do not.

The Dictionary contains both TA dependent words and TA independent words. Keeping in mind that the end result of EUROHELP is a Help System Development System, the Dictionary may be divided into a generic part which is part of the Help System Development System and contains TA independent words such as auxiliary verbs, determiners, and pronouns, and a TA dependent part, which the IHS developer will have to fill in with TA dependent words. This division would reduce the IHS developer's workload concerning generation of the dictionary.

3.3. The Interpretation Rules.

The Interpretation Rules are used to interpret information in the Application Model. For example in order to choose quantifiers and form of nouns. An example is that the indication "(Max 1)" shows that a concept cannot be described by a plural noun but is always in the singular.

3.4. The Linguistic Rules.

The Linguistic Rules control the linguistic realization of the list of templates into NL help texts. They consist of Syntax Rules and Cohesion Rules. The Syntax Rules are not further described in this paper.

The concept of cohesion is crucial for the quality of the help texts: The output from the Utterance Generator must form a unified whole and not just a sequence of unrelated sentences in order to be characterized as a text at all (Halliday and Hasan [3]. The Cohesion Rules are used to ensure that this requirement is fulfilled, i.e. they control the different means by which sentences may be connected.

The Utterance Generator achieves cohesion in the help texts via rules concerning:

- coordination of sentences
- ellipsis of subject and/or verb
- relativization
- pronominalization of subjects and objects.

The decision of which rules to use and when to use them depends on the number of templates making up the list, their type, and their topic.

For example: Even though the help texts in Figure 6 begin with the same sentence, pronominalization is used in example 2 to make the text less complex and more legible.

1. Delete is a command which marks one or several messages for deletion.

2. Delete is a command. It takes one or several messages as parameters and marks them for deletion.

Figure 6

Another example of a rule (which is obvious for human beings) is that relativization is enforced in example 1, which makes the text more fluent than if pronominalization was applied in this case - and keeps the system from generating ill-formed texts as "Delete is a command and marks one or several messages for deletion".

The precondition for actually applying ellipsis, relativization, or pronominalization is that the topic of the constituents in

question (e.g. the subjects) in two consecutive templates must be the same - as illustrated by the example*** below:

Delete is a command. Delete marks one or several messages for deletion.
|
V
Delete is a command which marks one or several messages for deletion.

Figure 7

3.5. The Generation Process.

The Utterance Generator generates the NL help texts from the list of templates by first applying the Cohesion Rules. The Cohesion Rules are applied first, before inserting NL words, since they do not operate on the actual NL word and since they may result in changes to what must be output (e.g. pronoun instead of noun)(in the following examples, the list of templates is presented as a tree-structure):

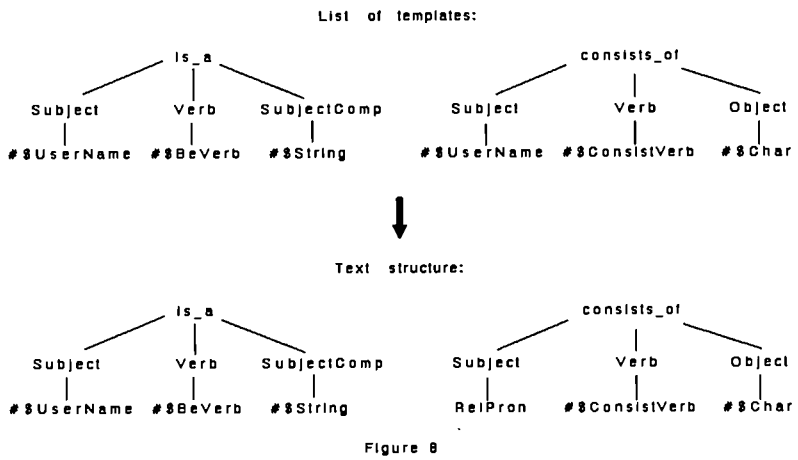


Figure 8

In this example, the indication "RelPron" in the subject of the second template shows that the Relativization rule has been applied. The surface form of the relative pronoun is inserted later in the generation process.

Then, Interpretation Rules and Syntax Rules are used during Dictionary look-up to make sure that correct forms of the NL words are inserted in the sentences, replacing the initial information selected from the Application Model:

*** This example is written in NL form instead of in its representation in the system in order to make it more legible.

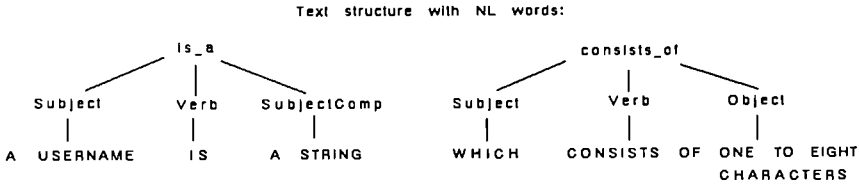


Figure 8

Finally, Linearization Rules are enforced:

A username is a string which consists of one to eight characters.

Figure 10

4. CONCLUDING REMARKS.

The prototype of the Utterance Generator generates NL help texts which are syntactically correct and coherent.

Because of the relatively limited Application Model used for the prototype, the NL sub-set handled currently is very restricted. For example, causal and temporal relations are not described in the Application Model and, consequently, the issues related with them (e.g. the choice of tense, adverbs etc.) are not included in any linguistic rules.

However, working with the prototype showed that the structure of the Utterance Generator is flexible - allowing modifications to parts of it without consequences for its structure as a whole. This facilitates future expansions of the NL sub-set and also allows adaption to other Indo-European languages than English.

As the description of the limitations on the Utterance Generator caused by the under-sized Application Model shows, future work on the Utterance Generator would benefit considerably from being integrated with "real-life" parts of the IHS. Therefore, the experience gained from the prototype of the Utterance Generator will be used in the forthcoming development of a pilot-system incorporating all components of the IHS.

[REFERENCES]

[1] Stausholm, L., Functional Specification of an Utterance Generator (CRI/EUROHELP/050, 1987).

[2] Rubinoff, R., Adapting MUMBLE: Experiences with Natural Language Generation (in Proceedings of AAAI, Univ. of Pennsylvania, Philadelphia, Pa., 1986, pp. 1063-68).

[3] Halliday, M.A.K. and Hasan, R., Cohesion in English (Longman, London, 1976).

BIBLIOGRAPHY.

Danlos, L., Generation Automatique de Textes en Langues Naturelles (Paris, 1985).

Holm, J., EuroHelp - Intelligent help systems. Experiences with a prototype and directions of future work (Esprit Technical Week, 1986).

Jacobs, P.S., Generation in a Natural Language Interface (in Proceedings of IJCAI, 1983, pp. 610-612).

McKeown, K.R., Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language (Cambridge University Press, 1985).

Quirk, R. and Greenbaum, S., A University Grammar of English (Longman, 1973).

Stausholm, L., Generation of NL Help Texts (CRI/EUROHELP/045, 1986).

Stausholm, L., Description and Evaluation of the Prototype of the Utterance Generator (CRI/EUROHELP/054, 1987).

Project No. 26

A Control Strategy for a Knowledge-Based Approach to Signal Understanding

Egidio Giachin, Claudio Rullent

CSELT - Centro Studi e Laboratori Telecomunicazioni
Via Reiss Romoli, 274 - 10148 Torino (Italy) - Tel. +39-11-21691

ABSTRACT

This paper pertains to the activity on the understanding level of a continuous speech understanding system. Two are the most important topics involved in this research: knowledge representation and control strategies. The speech understanding level deals with a lattice of word hypotheses instead of a sequence of words, thus the techniques for natural language understanding must deal with the preoblems caused by the uncertainty of the input data, i.e. very large search space and risk of erroneous interpretations.

The paper describes a control strategy characterized by integrating top-down and bottom-up steps in a strictly opportunistic way. During the analysis both goals and phrase hypotheses are generated. At each control cycle the item that has the best support from the lexical hypotheses is selected and a forward or backward step is performed according to its type. The strategy is formalized through the definition of a set of operators that are applied on deductive process instances (goals and phrase hypotheses); the operators also allow the join of two different deductive processes that have evolved independently (forward or backward).

1. INTRODUCTION

The final goal of a continuous speech understanding system is the generation of a representation of the utterance meaning, beside the recognition of the utterance words. From this representation a proper action can be taken in order to satisfy the needs of the user that interacts with the system (for instance by giving him an answer to a question). Both activities, recognition and understanding, have to be performed and should take advantage from the knowledge about words, language and domain. Recognition must use that knowledge as a source of constraints for word disambiguation while the understanding activity is entirely based on that knowledge and requires the same effort as in the case of written natural language understanding.

The majority of the speech understanding systems developed during the DARPA project were primarily involved in recognition while understanding was considered only a secondary goal. The techniques they used for knowledge representation (mainly semantic grammars) were of low complexity as knowledge about language and domain had to be used for both recognition and understanding. That resulted in a limitation of the potentialities of the natural language understanding activity.

In our approach syntactic and semantic knowledge is not used to add constraints to a recognition system: the issue is a natural language understanding stage that deals with the results given by a recognition stage reflecting the current state of the art that uses only acoustic and phonetic knowledge. More precisely the understanding stage accepts a lattice of scored word hypotheses instead of a word sequence and performs the final part of the recognition activity, jointly with the understanding activity.

The output of the recognition stage is a lattice of word hypotheses, each of them characterized by a score reflecting the goodness of the match (fig. 1). The number of word hypotheses has to be as high as necessary to contain the right hypotheses (i.e. those corresponding to the actually uttered words). If such requirement is not met the understanding stage has to interact with the recognition level to further analyze restricted segments of the utterance. Improvements in signal processing and pattern recognition techniques will reduce the number of wrong word hypotheses generated by the recognition stage and will improve the reliability of their scores. On the other hand, improvements in natural language understanding techniques (for instance in knowledge representation of syntax and semantics) can be exploited independently from the particular signal

This research has been partially supported by the EEC ESPRIT Project n. 26.

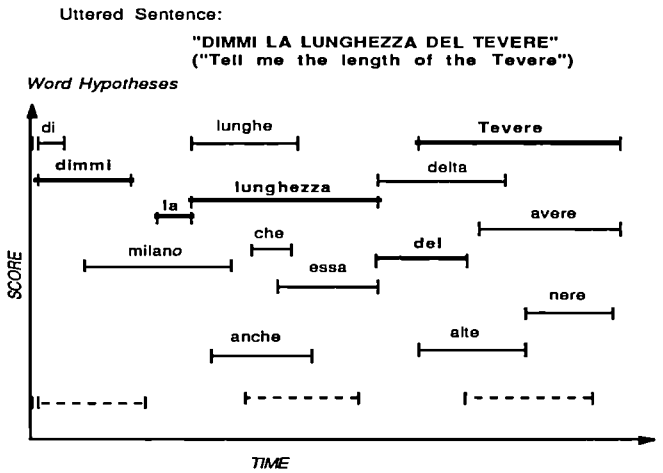


Fig. 1 - Structure of the lattice of word hypotheses.

processing techniques that are used. A real advantage of this approach is the possibility of using the same syntactic and semantic knowledge representations both as a source of constraints (to perform the final part of the recognition activity) and as a way of structuring word sequences in order to understand their meaning.

Nowadays the above mentioned approach is studied by a few reasearch groups, among them the group at Carnegie Mellon University [1] and the group at University of Erlangen-Nuernberg [2]. A common element of all these approaches is the use of a lattice of word hypotheses as the input of the understanding stage. These approaches to speech understanding are characterized by a more declarative way of representing syntactic and semantic knowledge with respect to the DARPA project (usually semantic grammars are not used any more) but in our opinion there are still some critical aspects that require new solutions. The final part of the introduction discusses two important aspects that have been analyzed during our research and whose solutions represent the innovative aspects of SUSY, the speech understanding parser that has been implemented in CSELT.

A convincing answer to the problem of an effective integration between syntactic knowledge and semantic knowledge is still to come. The problem is from one side to maintain independent and highly declarative representations for both semantic and syntactic knowledge and from the other to use them in an integrated way in order to exploit constraints as soon as possible. While this aspect is important for written natural language understanding (see [3] for a work of the authors on this subject), it is vital for speech, where the search space is very large, being the non-determinism of parsing added to the uncertainty of input data.

The central point of this paper does not concern knowledge representation, but control strategies for speech; anyway section 4 briefly describes the knowledge representation formalism adopted by SUSY. The reasons for the selection of such knowledge representation formalism, the importance of integrating syntax and semantics and finally the relationships between knowledge representation and control are described in [4].

While some speech understanding systems based on the use of semantic grammars, like HWIM [5], were really concerned about the problem of control, now this aspect seems to be underestimated. That is not surprising, as an increased complexity of the representation formalisms for syntax and semantics makes a formal control policy hard to reach and often induces to the use of heuristic methods.

The paper is organized as follows. Section 2 describes the basic requirements that a parser for continuous speech should have; section 3 gives an overview of SUSY. Section 5 describes the reasons for an effective formal control strategy in a problem solving environment and illustrates the difficulties that arise when such approach is taken. The solution of these incoming problems requires the integration of top down and bottom up activities through the generation of expectations. Refer to [6] for a more precise comparison of our system with the most similar systems like HWIM and HEARSAY-II and for a more precise description of our approach in terms of a highly controlled

blackboard system. Conceptual items called 'Deduction Instances' that represent intermediate steps of deductive processes are introduced in section 6 while a framework that allows an effective control strategy and the operators involved are outlined in section 7. Section 8 is devoted to a description of the memory structures that are used to represent phrase hypotheses. Finally, section 9 contains an example that should better clarify both the aspect of knowledge representation and the parsing strategies that have been previously described.

2. SOME BASIC REQUIREMENTS OF A PARSER FOR SPEECH

The understanding stage needs to detect, in the lattice, the best scoring sequence of word hypotheses covering the whole utterance and compatible with the models of the language and of the domain. The presence of word hypotheses spread all over the utterance, instead of a sequence of words, requires a parser whose main features are related to a very high flexibility in the control strategy. Some features of the parser are the following:

- It is important to have powerful control strategies based on the combination of word scores. An efficient parser must take this aspect into account.

- Due to the limitations inherent in the recognition stage, a "tolerant" parser is required:
  . Contiguous word hypotheses can slightly overlap, likewise gaps can exist between them.
  . Very short words (e.g. articles) are normally difficult to be detected by the recognition level and could be missing from the lattice; if they do not convey essential semantic information the parser should understand the sentence all the same.
  These requirements argue against a left-to-right parser.

- The parsing strategies must be suitable for parallelism. Only a highly parallel machine can perform speech understanding in real time. See [7] for a discussion of a possible way of exploiting parallelism from the parsing strategies adopted by our understanding stage.

- Syntactic and semantic knowledge must be separately defined and used in a joint way. Such separation allows a reduction of the time required for an expert to define an application for a new domain, i.e. to declare all the knowledge required to adapt the speech understanding system to the new domain.

3. OVERVIEW OF SUSY

A simplified overview of SUSY is reported in Fig. 2. A recognition level, that makes no use of syntactic and semantic knowledge, generates a lattice of scored word hypotheses that constitutes the input data of SUSY; see [8] for recent work on the recognition system. The uttered sentences are questions aimed to extract information form a data base pertaining to a given domain (Italian geography).

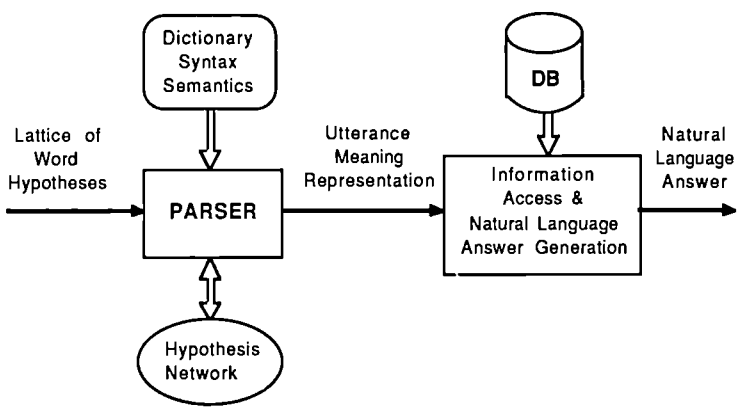


Fig. 2 - A simplified overall architecture of SUSY.

The lattice is processed by a parser that recognizes the most likely word sequence and generates an internal formal representation for the meaning of such word sequence. The formal representation, a conceptual graph of domain concepts connected by relations, is used to extract the required information from the data base. Starting from such information a natural language answer is generated and given to the user.

The parser is based on the use of a dictionary and on a set of syntactic and semantic rules. The internal structure generated and used by the parser is a network of phrase hypotheses. So the parser activity consists in generating continuously new phrase hypotheses that represent alternative or cooperative paths of the parsing activity.

4. KNOWLEDGE SOURCES FROM DEPENDENCY RULES AND CONCEPTUAL GRAPHS

The parser, during its activity, makes use of the following different kinds of knowledge:

- A dictionary, where each domain word is characterized by its morphologic and semantic features.
- A set of dependency rules (Dependency Grammar formalism, [9]) augmented with rules for controlling morphologic agreement conditions; dependency rules must constitute a subset of the language sufficient to cover the application.
- A set of caseframes expressed using the conceptual graphs formalism [10]. They describe domain knowledge and are represented by domain concepts connected by conceptual relations.

Starting from the last two knowledge bases and from additional syntax/semantics mapping knowledge, an integration of syntax and semantics is performed, generating items generically called 'Knowledge Sources' (KSs) in the remaining of the paper.

The parsing strategy can be better explained in terms of a blackboard system. The whole set of Knowledge Sources constitutes a deduction system: the actions that a KS can perform when it is triggered will be described in section 7 in terms of five operators; some of them behave in a forward fashion, some in a backward fashion while others allow integration of different deductive processes.

Each KS integrates different types of knowledge. First, the KS is characterized by a compositional structure that results from a 'compilation' process that integrates the semantics of one or more caseframes (expressed by conceptual graphs) with the structure of one or more dependency rules. This compilation process is performed off-line through the use of "mapping" information, that relates implicit grammatical relations of the dependency rules with the semantic relations of conceptual graphs (see [11] for a discussion about the utility of such mapping in the general case of natural language understanding).

A KS is also characterized by the associated dependency rules augmented by rules for the control of morphologic agreement. This knowledge is transformed off-line into special structures suitable to perform efficiently the activity of constraint propagation. All the other kinds of knowledge used by a KS are expressed through procedures. For instance a procedure (that makes use of thresholds) represents knowledge about the recognition system word-spotting characteristics; it is used to impose constraints on the allowed gaps and overlaps between word hypotheses.

Two Knowledge Source examples are shown in Fig. 3. Knowledge Sources have a compositional structure that is, in some way, similar to that of a rule, the basic difference being the fact that the AND premises are not at all independent but highly constrained by different kinds of knowledge: temporal constraints, morphologic and functional constraints, etc. It is the compositional structure of the KSs that permits to see them as constituting a Deduction System.

5. THE IMPORTANCE OF CONTROL STRATEGIES

The word lattice is usually characterized by a lot of spurious word hypotheses intermixed with the correct ones (i.e. those that correspond to words really uttered and covering the given time interval). Some spurious word hypotheses may also happen to have a better score than the correct ones.

5.1 Two good reasons for an effective control strategy

In a problem solving approach to speech understanding two main problems arise:

- There is a risk of erroneous understanding, that is spurious word hypotheses of the lattice can lead to incorrect solutions before the right one is found. So the utterance can be incorrectly understood.
- The search space is very large, adding the non-determinism typical of the parsing to the uncertainty of input data. The whole search space cannot be explicited.

The elimination of incorrect solutions requires a method for comparing solutions so that the best one

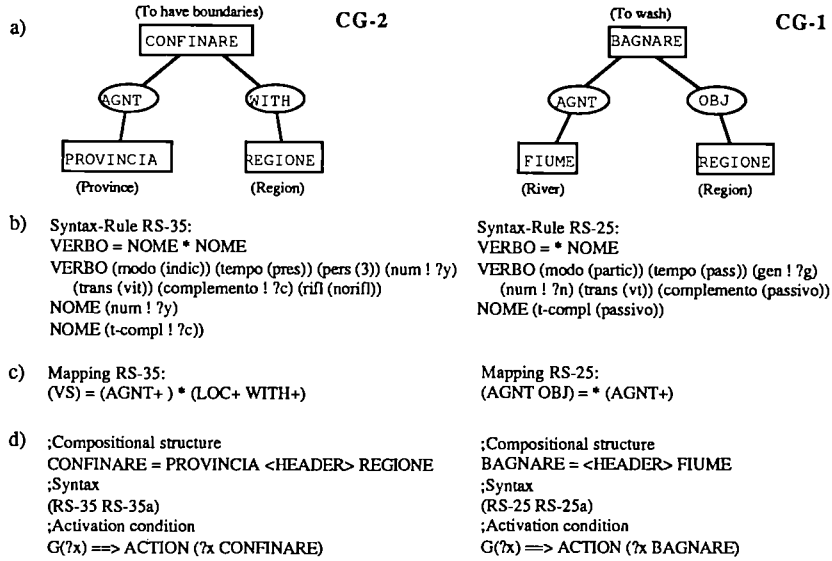


Fig. 3 - Simple KSs (d) derived from conceptual graphs (a), dependency rules (b) and mapping rules (c). In the KSs, the pointers indicated in the Syntax slot (RS-35, etc.) refer to data structures not shown in the figure.

can be selected as the correct one. A number, called Quality Factor, is assigned to them; this number depends only on the word hypotheses involved in the solution. So we assume that a formal probabilistic method can assign a number, called Quality Factor, to combinations of word hypotheses, starting from their scores and from their intervals. Probabilistic models of the source of word hypotheses and statistical correlations among them are not considered, due to the intractable complexity of the required methods. We have experimented the following ways of assigning Quality Factors:

- Joint probability: sum of the word hypotheses scores (seen as logarithms of their probabilities).
- Score density with or without shortfall [12].

In the examples a simplified view of the Score Density is considered: the average value of the scores of the word hypotheses involved.

As regards the second problem, Quality Factors must be used also to direct the search, in order to find the solution long before having to perform an impossible exhaustive search.

From the probabilistic point of view the most natural way of dealing with scored input data is to start with the best word hypotheses and trying to combine them together until a solution is obtained. The problem is the necessity of exploiting constraints from domain knowledge as soon as possible to drastically reduce the combinatorial activity. On the other hand a good way of exploiting domain constraints is a bottom up parsing strategy guided by the score of the word hypotheses in the lattice. But this approach is inadequate when the search space is very large due to a great amount of noise. In fact dangerous bottlenecks cannot be avoided if expectations are not considered. As an example consider a situation where a low level constituent, part of the solution, has to be formed using a very bad word hypothesis w1 (this could easily happen). The problem is that the solution can have a good Quality Factor (the bad score of w1 is compensated by the good scores of the other word hypotheses) but it can be unduly delayed by w1 because a lot of word hypotheses have to be considered whose score is better than w1 but worse than the Quality Factor of the solution.

5.2 The role of expectations: Integrating top down and bottom up parsing strategies.

An important feature of the parser in order to deal with these difficulties is the possibility of creating expectations at the highest levels. This is always possible in our approach because each KS

has been obtained from a caseframe and can be triggered by a word that represents the caseframe header. So a good word hypothesis has always a KS that can be activated by it. In addition the use of dependency rules perfectly suits to this point: each node of a dependency tree is taken by a word and such word hierarchy allows the generation of high level expectations.

The informal conclusion is that in some cases good word hypotheses cause the parsing to proceed bottom up while in other cases they first create expectations (goals) and then cause the parsing to proceed top down, looking for word hypotheses when necessary in order to perform single backward steps. The acquisition of a new word hypothesis during a top down step usually worsens the Quality Factor of a subgoal (incomplete phrase hypothesis), delaying its processing, while in the meantime other phrase hypotheses will be processed. Integration among bottom up and top down steps is vital: having to solve an incomplete phrase hypothesis a check is made to see if a suitable complete phrase hypothesis has already been generated and vice versa. The next section introduces conceptual items called 'Deduction Instances' that simplify the description of the parsing control strategies.

6. DEDUCTION INSTANCES AND SEARCH

The understanding of an utterance is completed when a solution S involving a certain set w1,...,wn of word hypotheses is obtained. The Quality Factor resulting from w1,...,wn is supposed to be the best one among the possible solutions. Such solution can be represented by a Deduction Tree: the AND tree whose nodes are facts (complete phrase hypotheses) and (sub)goals (incomplete phrase hypotheses). Following the informal guidelines of the previous section, a solution is obtained starting from one or more initial word hypotheses (the best ones) and then performing predictions, bottom up and top down steps and joining constituents.

Let us consider the simple case of a single initial word hypothesis that performs a prediction generating a goal that is solved through a sequence of top down steps. From the probabilistic point of view, new word hypotheses are connected one by one (assuming they satisfy all the required constraints) to the initial one until the solution is reached, connecting all the word hypotheses w1,...,wn. We call this activity a Deductive Process and each intermediate step is called Deduction Instance (DI). Some steps consist in adding a new word hypothesis to the existing ones, others represent only activities performed by a KS that do not involve the acceptance of a new word hypothesis. The OR alternatives of the overall search process are taken into account by different DIs. Each Deduction Instance can be represented by its Deduction Tree and it is characterized by a Quality Factor obtained applying a selected probabilistic method to the word hypotheses of the Deduction Tree.

A similar situation happens when bottom up steps are considered. In this case DIs have Deduction Trees whose nodes are all facts; they are called fact DIs while the others are called goal DIs. A single deductive process leading or not (if it fails) to a solution is a sequence of DIs.

6.1 Joining Deduction Instances

The optimal result would be obtained if the Quality Factors corresponding to the sequence of DIs worsen gradually in quality converging to the Quality Factor of the solution. The required integration among bottom up and top down steps can be obtained by merging together two Deductive Processes that have previously evolved independently: from two DIs a new DI is generated.

With some simplifications the whole deductive activity can be seen as a search in a state space. A state is a deductive process at a certain point of its evolution, i.e. a Deduction Instance. Operators can be applied on these states performing single prediction, bottom up, top down or merge steps. To each state a Quality Factor is also associated, then a best-first search can be performed; the state priority is given by the DI Quality Factor. On each state all the possible operators are applied. The next section presents the conceptual structure of the Control Strategy and describes the involved operators.

7. CONTROL STRATEGY AND OPERATORS

The control of the deductive activity is carried out by a Deduction Scheduler that at every cycle selects the best item among the remaining word hypotheses and the DIs (phrase hypotheses generated so far and inserted into a network called Hypothesis Network). All the items have a priority given by their Quality Factors (in the case of a DI) or by their scores (in the case of a word hypothesis); these scores have to be comparable with the Quality Factors of DIs. Each goal DI is also characterized by a Current Subgoal, selected among its unsolved subgoals. If the Deduction Scheduler selects a DI, the Deduction Cycle is entered, otherwise the Activation Cycle is

performed.

The Activation Cycle is executed when the best DI has a Quality Factor worse than the score of the best word hypothesis. In that case such word hypothesis is extracted form the lattice, and the ACTIVATION operator is applied, making predictions. Given a KS the operator decides if it can be triggered by the given word hypothesis; if so a DI is generated and inserted into the Hypotheses Network. Quality Factors are assigned to the new DIs. Conceptually this operator creates expectations.

In the Deduction Cycle the selected (i.e. the best) DI is given to the KSs. The activities performed by the KSs can be summarized by the following phases:

- Solution Test:
  The DI is tested to see if it is an acceptable solution; if so it is stored in the solution list. When the strategy is optimal the first extracted solution is guaranteed to be the best one, and the analysis can stop. Otherwise the analysis goes on until the available resources are consumed, and then the best solution in the solution list is selected.

- Problem solving actions:
  The actions that a KS performs when it is triggered can be described in an abstract way through five operators that represent the process of generating new hypotheses starting from others. The operators can be triggered either by a DI or by a Word Hypothesis; the characteristics of the triggering element define what operator is applicable. Each operator application represents an alternative continuation of the deductive process leading to the selected DI. The operators are:

  - SUBGOALING:
    It is chosen if the selected DI represents a goal. It performs a subgoaling operation on the Current Subgoal (the subgoal currently pursued) of that DI. Such subgoal must be a non terminal one (i.e. it cannot be solved directly by word hypotheses). The subgoal is decomposed according to the compositional structure of the KS.

  - VERIFY:
    If the Current Subgoal of the selected DI is a terminal subgoal, this subgoal is tried to be solved through a matching against the lattice of word hypotheses.

  - PREDICTION:
    If the selected DI is a fact, it predicts the goals having non-terminal subgoals solvable by this fact DI.

  - MERGE:
    The selected DI is merged with some other compatible DIs of the blackboard. Starting from two DIs (the selected one and the other) a new DI is created; so two deductive processes which have evolved independently from one another meet into a new deductive process. If the triggering DI is a fact, then the KS tries to merge such fact DI with compatible goal DIs having subgoals that can be satisfied by such fact. If the triggering DI is a goal, then the KS merges the current subgoal of such DI with compatible fact DIs.

- Subgoal Selection:
  For each new goal DI the Current Subgoal is selected applying a Subgoal Selection Function.

- Constraint Propagation:
  For each new DI the constraint propagation is performed. The constraints induced by the acquisition of new word hypothesis are propagated everywhere needed on the new DIs.

- Quality Factor Computation:
  A Quality Factor is computed for each new DI.

8. REPRESENTING DEDUCTION INSTANCES WITH MEMORY STRUCTURES

An aspect that has to be considered when representing DIs with memory structures is to reduce the amount of memory required and to properly organize the memory structures in order to simplify operators application (the MERGE operator, mainly). This section deals with these aspects and illustrates the Blackboard organization: a Hypothesis Network that makes use of two classes of

links. The section can be skipped without compromising the understanding of the example in Sec. 9.

The blackboard elements are conceptual structures previously called Deduction Instances (DIs). The most trivial way of implementing them would be the use of an explicit Deduction Tree for each of them, but to keep memory occupation within reasonable limits it is necessary to make DIs share common parts, if any. A natural choice is to use AND-OR trees; unfortunately, a problem arises when constraint propagation is required, as in our case: the AND-OR trees representation assumes the OR alternatives to be independent from one another, but that is not true if constraints propagation has to be taken into account. An example will clarify this statement.

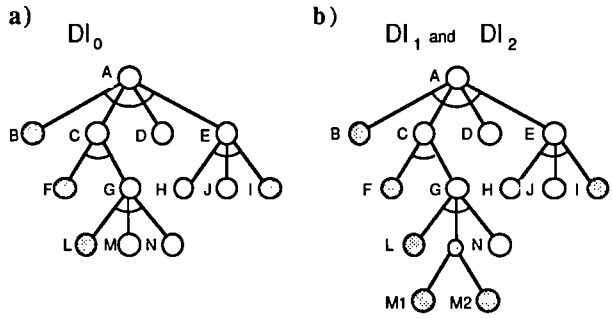


Fig. 4 - From a Deduction Instance DI0 (a), the resolution of subgoal M with two different word hypotheses generates two new Deduction Instances DI1 and DI2 (b), represented by an AND-OR tree: there will be problems due to constraints propagation.

Let us consider the goal Deduction Instance DI0 of Fig. 4-a, and suppose that the current subgoal M can be solved by two different word hypotheses generating two new DIs, DI1 and DI2 shown in Fig. 4-b and characterized by two competing solved subgoals (facts) M1 and M2. Now, suppose that N is selected as the current subgoal for DI1 or DI2. Since the two DIs are distinct and endowed with different word hypotheses, a word hypothesis satisfying N can happen to be compatible, say, with M1 but not with M2, and thus could be inserted in the context of DI1 but not in the context of DI2. This means that different constraints have to be propagated from the solved subgoals M1 and M2 to subgoal N, and N must be splitted in two subgoals N1 and N2, respectively associated with M1 and M2. By applying the same reasoning to any other subgoal in DI1 and DI2, it results that the whole Deduction Tree has to be duplicated, each of the new Deduction Trees having its own constraints. Trying to keep them implicitly united in a single structure would be of no use.

In order to still take advantage from the use of AND-OR trees also when constraint propagation has to be performed, a memory representation has been devised in which the nodes can be shared without n-plicating the tree. This is possible provided the topologies of Deduction Trees are constrained to have certain features. To obtain this result, limitations have been imposed on the ways deductive processes can go on; this is done by intervening during the current subgoal selection.

Such limitations do not compromise integration among top down and bottom up activities. The allowed tree topologies are called 'Canonical' and the resulting DIs are called 'Canonical DIs'. Let us introduce these concepts.

8.1 Canonical Deduction Instances (CDI)

We define CDIs starting from the definition of Canonical Deduction Trees (CDTs).

Definition 1:
- A DT is homogeneous if and only if it is a fact DT or a not yet decomposed (sub)goal.

Definition 2:
- A DT is canonical if it is homogeneous.
- A DT is canonical if:
  . All the (sub)DTs connected to the root are canonical and
  . No more than one of them is non-homogeneous.
- No other DTs are canonical.

Definition 3:
- A DI is Canonical (CDI) if and only if it corresponds to a Canonical DT.

From the definitions some consequences follow:

Proposition 1:
If a CDI corresponds to a homogeneous fact CDT, there is one-to-one correspondence between the CDI and the one-level AND tree whose root is the root of the associated CDT.
PROOF - Omitted (the statement is self-evident).
For reasons that we shall clarify in a short time, we call this tree the Representative of the CDI.

Proposition 2:
If a CDI corresponds to a non-homogeneous CDT, such tree contains exactly one non-homogeneous one-level AND subtree.
PROOF - By recursion: Consider the CDT associated to the CDI. If it is a one-level tree, the CDT itself is the subtree we looked for. Otherwise, since the CDI is canonical, Defs. 1 and 2 insure that its associated CDT has just one canonical non-homogeneous subtree. Then the above discussion can be applied to this subtree, until a one-level canonical non-homogeneous sub-n-tree is found; Q.E.D.

Proposition 2 implies that there is one-to-one correspondence between a goal CDI and such one-level non-homogeneous subtree. We call the subtree the Representative of the CDI (RS). For an example, see Fig. 5.

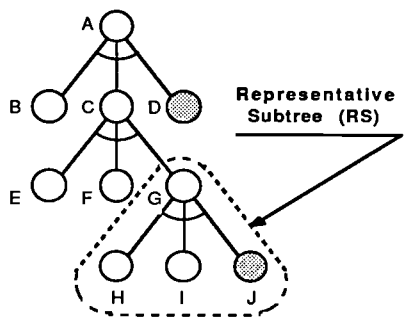


Fig. 5 - Representative Subtree in a Canonical Deduction Instance.

We give now the restriction on subgoal selection so that only canonical DIs are generated.

Subgoal Selection Rule:
The current subgoal of a goal CDI must be one of the goal leaves of the Representative Subtree RS.

The importance of CDIs lies in their one-to-one correspondence with their Representatives. In fact, to carry out the application of an operator on a CDI, the information provided by the Representative is sufficient. Thus we can use the Representatives instead of the whole CDI. Representatives are implemented by a memory structure called Physical Hypothesis, described in detail in the following.

8.2 Physical Hypotheses as representatives of CDIs

This section describes Physical Hypotheses and how they can represent CDIs in a fashion compatible with the use of AND-OR trees. A Physical Hypothesis (PH) is a memory structure that implements a one-level subtree. Physical Hypotheses stand for the Representative Subtrees of CDIs. They store all the information that is necessary to process a CDI when it is selected by the Scheduler. Similarly, when a new CDI is generated, only one new PH is created, representing the whole new CDI for future processing. If a PH represents a fact CDI it is said to be *complete;* if it represents a goal CDI it is said to be *incomplete.*

Physical Hypotheses in the Hypothesis Network are connected by Compositional Links to form AND-OR trees, called PH-trees. Every PH in the PH-tree represents a DI that constitutes the

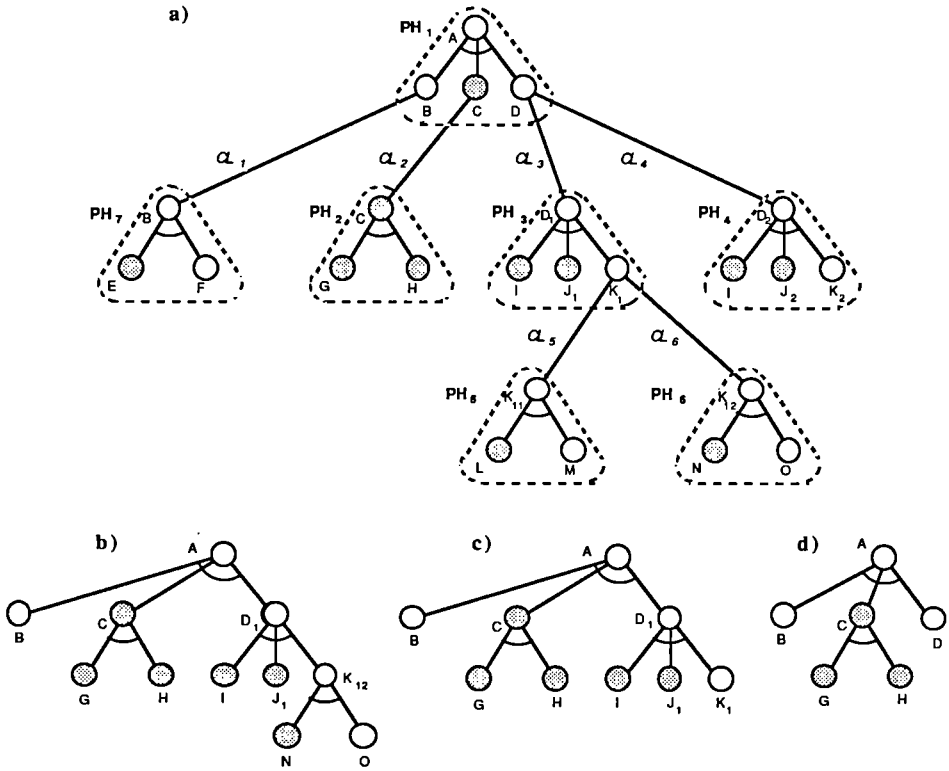


Fig. 6 - Physical Hypotheses and AND-OR trees. Solved subgoals are represented by shaded circles, then complete Physical Hypotheses have all their circles shaded.
a) The AND-OR tree of Physical Hypotheses represents 7 canonical DIs. 7 AND trees can be extracted, each corresponding to one canonical DI.
b) The Canonical Deduction Tree (CDT) corresponding to PH6. CL1 has been discarded, otherwise the DT would not have been canonical. When extracting AND trees, the CLs are taken into consideration if and only if they do not compromise canonicity.
c) The CDT corresponding to PH3. CL5, CL6 and CL1 have been discarded.
d) The CDT corresponding to PH1. CL2 has not been discarded: CLs to facts do not compromise canonicity and hence they are always considered.

only AND tree, extracted from the AND-OR PH-tree, that includes such PH and that has a canonical structure (remember Proposition 2).

PH-trees can have non-canonical structures, possibly with OR alternatives. Informally, one could say that a PH "sees" the PH-tree it is part of as lacking the PH-subtrees that would give rise to a non-canonical structure. Thus, no conflict will arise from the non-canonicity of the PH-trees. An example is shown in Fig. 6, where an AND-OR tree of seven PHs represents seven CDIs, three of which are depicted in the figure. The OR alternatives are independently interpreted thanks to the canonicity of the DIs. Summarizing:

- An AND-OR tree of n PHs represents exactly n CDIs.

- An incomplete PHi that is part of a PH-tree represents the Canonical Deduction Tree corresponding to the only AND tree, extracted from the AND-OR PH-tree, that is canonical and has PHi as its Representative Subtree (see Fig. 6). The PH represents a goal CDI and contains goal specifications and fact descriptions about the leaves facts/goals involved in the Reprentative Subgoal of the CDI. Summary information is provided for the root of the Representative Subgoal.

- A complete PHj (part of a PH-tree) represents the homogeneous Canonical Deduction Tree extracted from the AND-OR PH-tree, and having root PHj. The CDI associated to the CDT is a fact CDI. The PH represents a fact CDI and has the same structure of an incomplete PH, the difference being that there are no subgoals.

- When a subgoal is solved, a new PH is created in which constraints from the solved goal have been propagated to all of the remaining subgoals of the PH.

8.3 Specialization Trees

Since a PH represents a decomposition of a concept into subconcepts, it corresponds directly to a KS. Thus PHs can be grouped according to the KS they correspond to. Now, suppose to start from a Deduction Instance CDI1 and to apply an operator to it obtaining CDI2. CDI2 is more "specialized" than CDI1 because it has acquired new pieces of evidences or because its current subgoal has become more constrained, etc. When the PHs representing CDI1 and CDI2 refer to the same KS, PH2 (representing CDI2) is connected through a Specialization Link to PH1 (representing CDI1).

Specialization Links allow the generation of Specialization Trees. Each Specialization Tree corresponds to a KS and the level of a PH in the Specialization Tree corresponds to the level of completion of its Representative Subtree. The presence of Specialization Links connecting PHs, together with the Compositional Links that constitutes the AND-OR trees, explains why the working memory has been referred to as a Hypothesis Network. The links are frequently used during the analysis, mainly during the application of the MERGE operator to make easy the search for joinable DIs.

8.4 The MERGE operator

We consider now, as an example, the application of the MERGE operator. Given the current Deduction Instance CDI1, the MERGE operator tries to use other CDIs contained in the Hypothesis Network. Let be CDI2 one of the CDIs satisfying all the conditions for merging with CDI1. Then the MERGE operator applied on CDI1 and CDI2 generates a new Deduction Instance CDI3.

Fig. 7 describes the MERGE operator when one CDI is a fact and the other is a goal. Let CDI1 be the fact DI (represented by PH1) and CDI2 the goal DI (represented by PH2). The current subgoal CS of CDI2 has to be of the same type (D in figure) of the fact constituting CDI1. A new goal CDI3 is generated, substituting subgoal CS of CDI2 with the given fact. CDI3 is represented

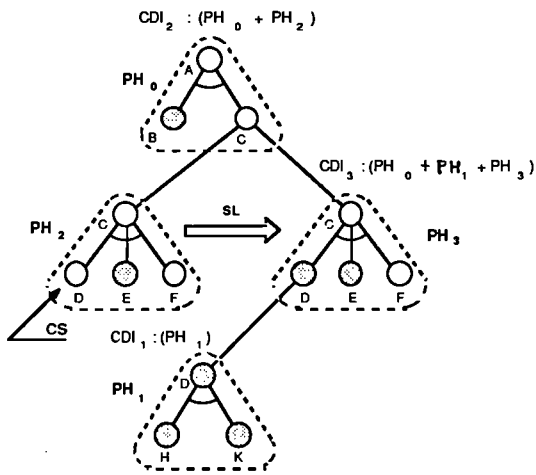


Fig. 7 - Merging a fact DI with a goal DI. The MERGE operator applied on CDI1 and CDI2 (represented by PH1 and PH2) generates CDI3 (represented by PH3).

by the Physical Hypothesis PH3. Since PH3 refers to the same KS as PH2 and CDI3 is more specialized than CDI2, PH3 is connected to PH2 through the Specialization Link SL.

9. THE EXAMPLE

The uttered sentence is reported in Fig. 8-b; for clarity, articles and prepositions ("dalla", "con", "le", "del") are supposed not to be in the lattice. The parsing algorithms are able to understand an utterance even if such word hypotheses are missing; nevertheless they are searched for, but phrase hypotheses are completed all the same even if they are not found.

Seven correct word hypotheses (small dark circles in Fig. 8-c) are in the lattice, intermixed with noisy word hypotheses, not reported in figure. The vertical position of the initial word hypotheses and of the generated phrase hypotheses corresponds to their score/Quality Factor (QF). The horizontal position of word hypotheses and DIs corresponds to the position inside the utterance time interval and is graphically centered on the word of the utterance word sequence in Fig. 8-b that corresponds to the head node of the Representative Subtree of the DIs (or to the word hypotheses itself). In the example QFs are computed using a simplified density method: when word hypotheses are combined, the resulting QF is the average value of their scores (in other words all the time intervals of the word hypotheses have been assumed to be 1).

Fig. 8-a represents the final representation of the utterance meaning: a conceptual graph resulting from the join of the two canonical graphs represented in Fig. 3 together with a third one that relates a province to a region through a "part-of" relation. In addition a few dependency rules are outlined; refer back to Fig. 3 for an example of two complete dependency rules involved in the example and of their associated mapping rules.

The following describes the parsing phases relevant for the solution (i.e. the parsing activities that are related to noisy data are not described):

- The best scored word hypothesis (see Fig. 8-a and -c), "regioni" (regions), activates a Knowledge Source (KS) that results from the combination of dr-1 (see back Fig. 3, rule RS-25) with a set of conceptual graphs that includes cg-1. Such word plays at the same time the role of governor of the dependency rule and the role of the "obj" case filler of the cg-1 caseframe. As such it satisfies the activation condition of the KS and the Deduction Instance d1 is then generated.

- d1 is still the best scored item : the SUBGOALING operator is applied and a set of new KSs are triggered, among them a KS whose activation condition requires a VERB and a concept like BAGNARE (to wash), ATTRAVERSARE (to cross), etc. The VERIFY operator is immediately applied and, thanks to the word hypothesis "bagnate", the caseframe header (whose "obj" case was already filled) is obtained and an incomplete phrase hypothesis d2 is generated.

- But d2 is not the best item any more: the word hypothesis "quali" (which) has a better score and it is activated, generating d3; "quali" is an interrogative adjective that can be taken into account by the dependency rule dr-4. Now a prediction activity is performed: a KS that has been obtained from dr-4 is triggered and the VERIFY operator takes into account the very bad word hypothesis "province", generating d4.

- Given the bad QF of d4, this inferential process is suspended. At this point d2 is resumed and a further top down step is performed, generating the complete phrase hypothesis d5, thanks to the word hypothesis "Tevere" that fills in the remaining "agnt" case of the "bagnare" (to wash) caseframe.

- d5 has the best QF among the active items and a prediction activity is performed (perhaps after some other noisy items have been processed). Among the others a KS resulting form dr-3 and a concept like CONFINARE (to border on), is triggered by d5, by filling a "with" case; the result of the VERIFY operator is d6, where the word hypothesis "confinano" constitutes the header.

- The word hypothesis "Campania" has a better QF than d6 and is then activated generating a complete phrase hypothesis d7 that is joined with d4 generating the complete phrase hypothesis d8. At this point d6 returns to be the best item and a top down step is performed: the "agnt" case is filled trough a join with d8, generating the solution d9 with QF = .486.

10. CONCLUSIONS

The basic consideration that has led us to this new parsing strategy for continuous speech can be summarized in the following way: if a recognition level has a good performance, then the number of incorrect word hypotheses with score better than the Quality Factor (QF) of the solution should not be too large, while it is possible to have a small number of correct word hypotheses

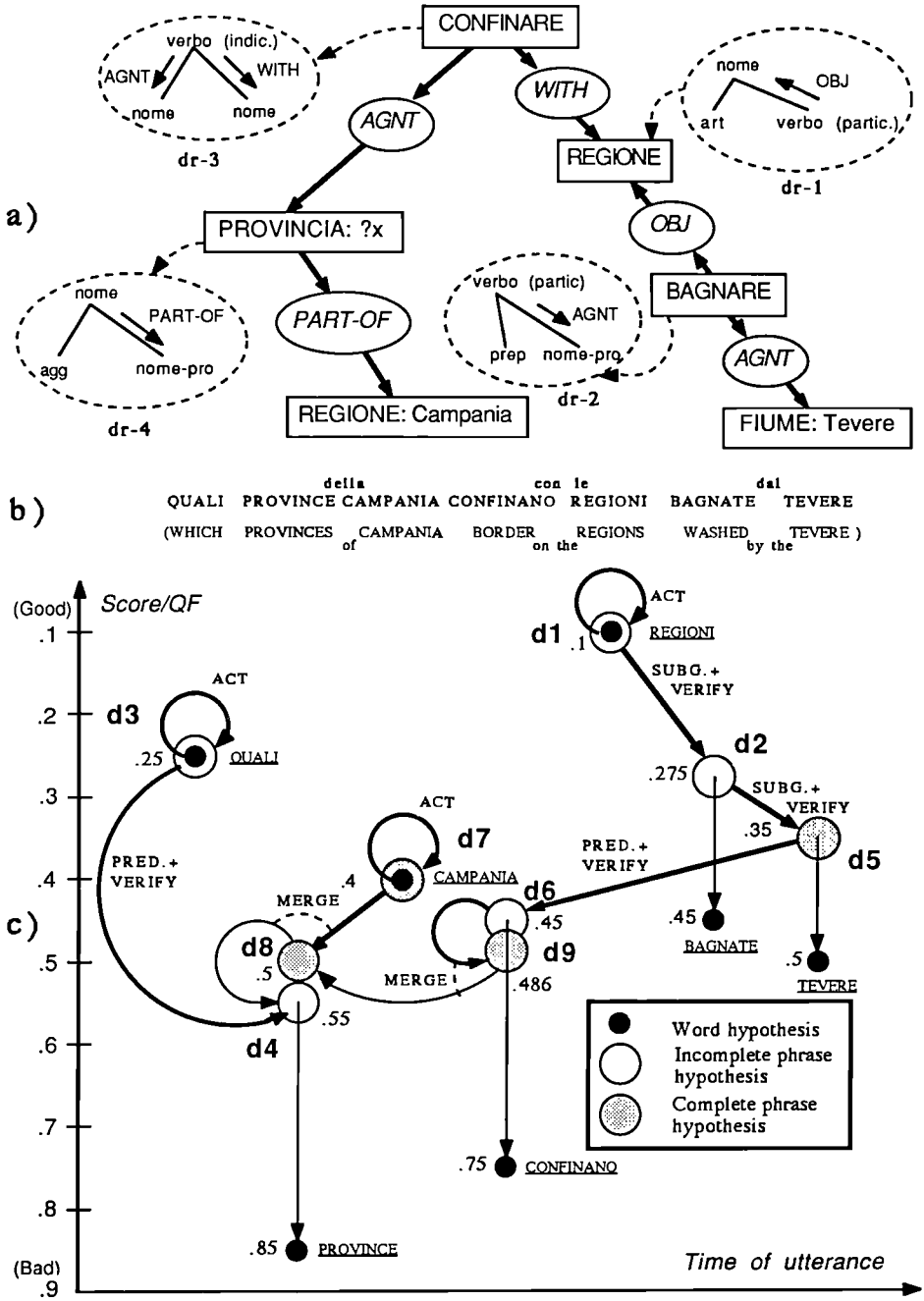


Fig. 8 - A parsing example: a) The resulting meaning representation together with a few dependency rules. b) The uttered sentence and its literal translation. c) The initial word hypotheses and the generated DIs in a time vs. score plane.

with low scores, for instance due to bursts of noise, imprecise probabilistic models of some words, etc. These hypotheses are usually intermixed in the lattice with a lot of bad and incorrect word hypotheses; we call bad word hypotheses those that have a score worse than the solution's QF.

The main purpose of our control strategy is to avoid the need of combining together large quantities of incorrect word hypotheses with bad scores while still being able to "capture" among them the correct ones. The described parsing strategy and careful constraint propagation algorithms make possible to combine only those bad word hypotheses that can be part of specific, well constrained contexts; that results in a drastic reduction of the number of phrase hypotheses supported by bad word hypotheses.

The example shows that the parser is able to start parsing processes anywhere in the time interval and, given the best parsing process, always has the chance of making a single step towards the solution and to suspend it if that is required by the QF of the Deduction Instance: there are not fixed parsing directions at any instant during the parsing activity. Efficient parsing techniques have been developed to allow the confluence of two parsing processes (remember the Specialization Trees described in section 8 for the application of the MERGE operator).

From the knowledge representation point of view the most suitable formalisms have been selected (Dependency Grammars for syntax and Conceptual Graphs for semantics) and combined into Knowledge Sources that allow the creation of expectations at all the levels. The combination of syntax and semantics into KSs has also the basic advantage of better exploiting syntactic, morphologic and semantic constraints during the parsing activity.

The proposed control strategy is not limited to speech, but it is useful whenever a problem solving approach is taken to deal with uncertain data. The only requirement is the possibility, for the KSs, of creating expectations at the highest levels.

REFERENCES

[1] Hayes,P.J., Hauptmann,A.G., Carbonell,J.G., Tomita,M., 1986. "Parsing Spoken Language: a Semantic Caseframe Approach", *Proc. COLING-86*, Bonn.

[2] Brietzmann,A., Ehrlich,U., 1986. "The role of semantic processing in an automatic speech understanding system", *Proc. COLING-86*, Bonn.

[3] Comino,R., Gemello,R., Guida,G., Rullent,C., Sisto,L., Somalvico,M. 1983. "Understanding Natural Language through parallel processing of syntactic and semantic knowledge: an application to data base query." *Proc. 8th IJCAI,* pp. 663-667.

[4] Poesio,M., Rullent,C., 1987. "Modified Caseframe Parsing for Speech Understanding Systems", *Proc. 10th IJCAI,* Milano.

[5] Woods,W.A. et al., 1976. "Speech Understanding Systems: Final Report." Rep. No. 3438, BBN, Cambridge (MA).

[6] Gemello,R., Giachin,E., Rullent,C., 1987. "A Knowledge-Based Framework for Effective Probabilistic Control Strategies in Signal Understanding.", *Proc. GWAI 1987.*

[7] Bosco, P.G., Giachin, E., Giandonato, G., Martinengo, G., Rullent, C., 1987. "A Parallel Architecture for Signal Understanding through Inference on Uncertain Data", *Proc. of PARLE - Parallel Architectures and Languages Europe*, Eindhoven (NL), in "Lecture Notes in Computer Science", vol. 258, pp. 86-102.

[8] Laface, P., Micca, G., Pieraccini, R., 1987. "Experimental results on a large lexicon access task", *Proc. ICASSP-87*, Dallas.

[9] Hays D.G.,1964. "Dependency theory: a formalism and some observations", Memorandum RM4087 P.R., The Rand Corporation.

[10] Sowa, J.F., 1984. "Conceptual Structures", Addison-Wesley, Reading (MA).

[11] Danieli, M., Ferrara, F., Gemello, R., Rullent, C., 1987. "Integrating Semantics and Flexible Syntax by Exploiting Isomorphism Between Grammatical and Semantic Relations.", *Proc. 3rd Conference of the European Chapter of the ACL*, Copenhagen.

[12] Woods, W. A., 1982. "Optimal Search Strategies for Speech Understanding Control.", *Artificial Intelligence* 18, pp. 295-326.

Project No. 940

STEREO RECONSTRUCTION USING A ROBOT MANIPULATING ARM

G. Garibotto (ELSAG)
Tech. Coord. ESPRIT Project P940
Participants: Cambridge Un., ELSAG S.p.A., Genoa Un., G.E.C., INRIA, ITMI, MATRA S.A., NOESIS

Abstract

The paper describes, in a qualitative form, the recent results obtained in the ESPRIT Project P940, entitled *Depth and Motion Analysis*. In particular the presentation will be limited to 3-D stereo reconstruction which has proved to be the most established topic of research within the project. The proposed stereo approach consists in a trinocular configuration using edge features which are clustered into linear segments and used for stereo correspondence. In this way it is possible to achieve a first order average of the edge point positions, and minimize noise effects in 3-D reconstruction. Preliminar results are referred in the paper using a robot manipulating arm in a rather simple configuration. Increased flexibility will be demonstrated at the Conference, by using eye-in-hand scene acquisition.

1 INTRODUCTION

The main purpose of the research project P940 is the study and realization of a prototype system, to recover 3-D data from stereo and motion information [1]. This processing system will be integrated in two different

application domains: a mobile vehicle to be able to orient itself and build 3-D visual maps in indoor scenes and a robot arm for object classification and manipulation.

In the first case the typical environment is that of an office, with ordinary furniture, doors, windows, corridors, desks with computer terminals, books and telephone sets, file cabinets, etc. In order to allow the system to move friendly within such environment and to be able to recognize them accordingly, an accurate 3-D description of the scene is required mainly in the form of planar surfaces. An efficient way to display the results seems to be the projection of the detected 3-D features in three orthogonal views.

In the second case the robot manipulator is supposed to be able to recognize and move to the appropriate position a set of *simple shaped* objects, like cylinders, cones, spheres, for which a reference model will be available. Moreover, complex objects from industrial environment will be handled, in order to automatically find the most suitable grasping position, starting from their convex hull description, through a more accurate surface representation in terms of high level primitives.

In both demonstrators the optical axes of the three TV cameras can be arbitrarily oriented to a common fixation point, and appropriate calibration procedures are required to correctly compute such orientation. Moreover, this calibration parameters are also used to perform epipolar transformation of the significant edge features, and simplify feature matching.

The paper summarizes the major results obtained in our project, as far as passive stereovision process is concerned, with no discussion on other fundamental topics of our research such as motion analysis, 3-D description and representation problems. Further details on these subjects can be found in the official deliverables of the project and related publications which are quoted in the references.

Section 2 will describe the full stereovision process, providing a synthesis of the investigation carried out so far within the project and the major achievements obtained from this analysis. Section 3 will refer on edge detection techniques, to extract the required tokens for stereo correspondence. Section 4 is then devoted to discuss the problem of camera calibration and the obtained results have proved to be very promising in our applications. Stereo matching and epipolar transformation are then discussed in section

5 and an example of segment matching is referred in section 6, together with a description of the robot system configuration which is used for data acquisition and scene inspection. Finally, section 7 will introduce some of the advanced topics of research which are currently investigated in our project, and summarize the basic strategy which has been adopted by our team.

2 STEREO APPROACH

Our consortium has selected edge points as the appropriate tokens for both Depth and Motion analysis. So far we have mainly investigated and used step edge models, although significant improvements are epected from including other models like roof edges, corners, occluding edges, coloured edges, shadows, etc. Textures and regions, for different reasons, have not been found adequate to the requirements of our demonstrators.

Three different levels of token clustering have been considered, that is individual edge points, along epipolar lines, edge chains, with continuity and propagation constraints, and disjoint contiguous linear segments. It is well known that small errors in the estimation of disparity values determines significant fluctuations in the depth values so that it is extremely important to achieve subpixel accuracy in feature localization. Since the *best* edge detection process can achieve accuracy in edge localization up to the available spatial resolution, the only way to improve such precision is an averaging process, to be performed on connected edge chains.

Polygonal approximation represents an efficient first order linear smoothing of the contours into contiguous disjoint segments, so that the uncertainty associated to each individual edge point is suitably reduced. This token description is very appropriate for indoor scene representation, as required by the mobile vehicle application, due to the inherently linear structure of the environment [7]. In the representation of man-made objects with complex curved surfaces (industrial scenes) this solution is slightly less efficient in data compression, and the sparseness of 3-D reconstructed segments is sometimes unsatisfactory. On the other hand, since the recognition and manipulation tasks of the robot arm do not always require video-rate performance, some additional computation and a more accurate contour

description can be acceptable for this application.

Improved results are expected including a logical connection of 3-D segments coming from the same edge chain in the image plane. Continuity constraints along edge contours can be included also using individual edge point correspondence, as a final consistency check on the estimated disparity values. An alternative, still under investigation, consists in using edge chains as matching features, to include on-line constraints during the estimation of the disparity values. In this second case a chain scanning would be required, instead of the simpler row scanning, after epipolar line rectification.

A trinocular vision system has been adopted within the project for stereo reconstruction, to achieve at least two complementary goals. First, to increase sensitivity to different edge orientations, and minimize the ambiguity due to features oriented along epipolar lines. Moreover it provides an intrinsic triangular geometric constraint for edge matching, so that for any admissible correspondence between two different views, a similar edge feature should be found in a certain position (scaled triangle) in the third image, due to the epipolar geometry. Using this constraint most ambiguous matching are immediately removed at the very beginning.

The adopted dissimilarity function is essentially based on the magnitude of the gradient as a local attribute. Other functions like gradient slope and average intensity values have been also investigated in the project but they have shown less discriminant power.

In any case epipolar transformation has been always used to compensate for the convergency of the optical axes, and to obtain a more convenient arrangement of the conjugated epipolar lines along one of the principal axes on the image plane.

A further problem which has been taken into account is the optimum addressing of edge pixels for stereo matching. On one side finding the potential correspondences of edge contour points (both individual and along chains) is a relatively simple task due to their discrete spatial position (x, y) on the rectangular sampling grid. On the other hand, bucketing techniques are necessary when dealing with linear segment correspondence, since the end points are no longer quantized to discrete values.

3 EDGE DETECTION

As already mentioned, edge features have been selected for stereo matching. Edge detection is performed using the approach proposed by Canny [2], to maximize localization properties and S/N response, and minimize the effect of multiple responses to noise input. Edge contours are obtained by hysteresis thresholding, using two threshold values $S_1 < S_2$.

We have investigated some alternatives by considering both first and second spatial derivatives of the images, at various resolution. The gradient based approach has been finally selected, with respect to zero crossing, for many reasons, and in particular because it provides less distortion of straight segments around intersection points and it exhibits less sensitivity to noise, which allows to perform edge detection at higher resolution.

Different smoothing regularization functions have been considered by comparing gaussian functions against optimal operators for step edge models. A very efficient recursive implementation [3] has been proposed in our project, to allow a fixed low number of operations per output pixel (26 for gradient convolution) irrespective of the spread of the impulse response. Anyway, present technology suggests to use available building blocks for FIR implementation of the convolution filters. In our examples a hardware convolver [4] has been used to implement FIR filters and to compare the performance of two different gradient functions obtained as a truncated approximation to the gaussian derivative and the exponential function [3]

$$f'(x) = -xe^{-a|x|} \tag{1}$$

Using the performance criteria suggested by Canny [2], we obtain the results of table 1 where gaussian derivative is poorer in edge localization but is superior in terms of multiple responses. Furthermore, from a very practical standpoint, quantization effects on the image grid are so severe that no perceivable difference has been found in the considered examples, which seems to dictate that the shape of the regularization function is not so critical in finite word length FIR implementation.

Once the gradient function has been computed it is necessary to locally remove non-maxima samples, to come to connected groups of edge points of thickness equal to one pixel. The output edge map contains only those

size	$\Sigma\lambda$ gauss	Σ gauss	λ gauss	k gauss	$\Sigma\lambda$ deriche	Σ deriche	λ deriche	k deriche
5	0.609	0.712	0.856	2.585	0.704	0.707	0.996	0.192
7	0.735	0.818	0.898	0.795	0.784	0.796	0.985	0.256
9	0.843	0.936	0.846	0.571	0.909	0.929	0.978	0.293
11	0.873	1.148	0.761	0.546	1.022	1.046	0.977	0.324
13	0.889	1.282	0.693	0.540	1.140	1.167	0.977	0.345
15	0.901	1.399	0.644	0.530	1.246	1.281	0.972	0.362
17	0.907	1.495	0.606	0.540	1.336	1.384	0.965	0.376
19	0.905	1.592	0.569	0.528	1.416	1.481	0.956	0.388
21	0.920	1.684	0.546	0.543	1.487	1.575	0.944	0.397
23	0.919	1.775	0.518	0.535	1.545	1.660	0.930	0.404
25	0.896	1.850	0.484	0.529	1.597	1.744	0.916	0.410
27	0.898	1.929	0.466	0.540	1.644	1.824	0.901	0.414
29	0.917	2.009	0.457	0.549	1.677	1.897	0.884	0.420
31	0.932	2.058	0.453	0.555	1.708	1.965	0.869	0.424

Table 1. Comparison of performance of the FIR filter implementation of gaussian derivative with respect to the exponential function, using Canny's parameters (Σ, λ, k) for different truncated window sizes (from 5 to 31 taps).

samples having a gradient magnitude above threshold S_1. Quantization effects have been considered also in this operation of non-maxima suppression. As a matter of fact, it has been proved that 8-bit precision of the gradient function is not always sufficient, due to saturation effects.

Furthermore, edge pixels have to be connected into contour chains, as required by our stereo matching procedure. Different approaches to this problem have been investigated in our consortium and the adopted solution [5] has been selected mainly on the basis of hardware implementation constraints. It is performed in two steps: the first one consists in a 3×3 neighbour analysis of the edge samples, to collect together the connected pixels into a number of labelled lists. The second step is a list handling process to carry on hysteresis thresholding on the selected terms by keeping only those lists which contain at least one sample with local contrast above threshold S_2. Moreover, these lists are arranged into ordered edge chains which contain the spatial position of the edge samples (x, y) and their contrast attribute (magnitude of the gradient).

Next, to obtain a linear segment description of the contour, an algorithm for polygonal approximation [6] is used. Clusters of edge points which are well approximated by a line segment, in the mean square sense, are grouped and represented by the line equation. Each segment is characterized by its endpoints, its length, orientation and contrast, as the average contrast of the individual edge points.

Furthermore, to simplify the addressing problem in stereo matching, the neighbourhood structure of the set of line segments is made explicit, using the technique of bucketing [7]. The image is divided into a number of square windows; to each window is associated the list of line segments intersecting it and to each line segment is associated the list of square windows which are intersected by the segment.

4 TV CAMERA CALIBRATION

This operation is required by the configuration of the acquisition system where the optical axes of the cameras are arbitrarily oriented with respect to each other. The precision of these calibration parameters may significantly affect the following process of stereo correspondence. Two main

alternatives have been developed and investigated in our project. and are both essentially based on the estimation of the parameters of rigid motion according to different approaches, which are briefly recalled in the following.

4.1 Multiple views of a grid patterns.

The first approach consists in the estimation of parameters of rigid motion by looking at a known regular grid pattern. The basic technique consists in recording at least two different views of this pattern by setting a fixed known displacement of the imaging system, so that to put in correspondence two sets of non-coplanar points. Using the constraints of rigid motion it is possible to estimate the translation and rotation of the TV camera with respect to a reference system fixed to the pattern itself. The calibration algorithm is fully explained in [8], where Kalman filtering is also used to reduce uncertainty effects in the localization of the intersecting grid points in the image planes. In this approach each camera is calibrated with respect to the reference system, including its intrinsic parameters like the focal length, vertical v.s. horizontal aspect ratio, distortion parameters, etc. Satisfactory results have been found in the mobile vehicle application, with an estimated angular accuracy of less than one degree and half centimeter in the localization of the calibration grid. Further experiments are planned using a robot manipulating arm where the controlled movements are expected to be more accurate and reliable.

4.2 Use of vanishing points.

In this second approach the calibration process is decoupled in two steps. The intrinsic parameters of the cameras are estimated once for all using off-line geometric and mechanical tests. The spatial position of the imaging system is computed again with respect to a reference pattern, using the well known properties of vanishing points in perspective geometry. A set of parallel and orthogonal lines on a planar surface represent the calibration pattern and the perspective distortion from a slant view allows to estimate the position of the vanishing points in the image plane [9]. Henceforth, from a single registration is possible to recover the necessary rotation and

translation parameters of the TV camera. The precision of this method has been experimentally tested in computing the length of a given line on a table, and the obtained precision has been of less than $0.5cm$ over a length of more than $30cm.$, irrespective of the line orientation. Main advantages of this approach are the simplicity of the method and the single view requirement for updating the relative position of the camera with respect to the environment.

Further experiments are planned to verify the most efficient solution for the project; anyway a standard procedure has been developed to make it possible to use either one of the methods and share the obtained results within our consortium.

5 STEREO MATCHING

Different stereo matching algorithms have been investigated during the first year of our project.

On one side individual edge point matching have been considered along conjugated epipolar lines using a two camera system. In this case similarity measures were based on local gradient magnitude, orientation, local intensity values. Weak ordering constraints have been also introduced, to make it possible to recover very difficult situations of inversion and occluded edges [10].

On the other hand trinocular vision have been experimented using a special arrangement with parallel optical axes, using a translation of the camera to the three vertices of a rectangular triangle. In this way all the possible orientations of the edge features could be successfully handled, increasing the number of edge point matching along epipolar conjugated lines (either horizontal or vertical) [11].

Another approach was based on using line segments as tokens to be matched, so that reducing the number of tokens involved (data compression). A first method was based on two cameras and used a strategy of hypothesis prediction and verification in which each hypothesis of correct matching was based on the descriptive features of each line segment, position, length, orientation, contrast. Each hypothesis is then verified by propagating the initial match to the neighbours of the segments, using the

bucketing technique [10]

Further improvements have been obtained using a trinocular system where the third camera was used to verify the potential matches [7]. In fact, according to the triangular constraint of such configuration, any correct match should be verified in the three rectified images, by suitably scaling the reference triangle made by the optical centers of the three cameras (see fig.1). This geometric constraint has proved to be very powerful so that to overcome most of the matching problems found in a two camera configuration.

The selected solution is essentially based on this last results with some relevant improvements to deal with complex curvilinear shapes. In particular there will be no more a single master couple of views, with the third image simply used for consistency check. In fact, there is no reason to privilege one orientation of the edge features with respect to the others. Hence, the reference image will be coupled alternatively to one of the other two, depending on the local orientation of the considered segment to increase both density and accuracy of stereo matching. Moreover continuity constraints along edge chains will be considered to correctly propagate matching hypothesis and recover broken segment correspondence.

5.1 Epipolar transformation

As already mentioned, the general arrangement of the three cameras is supposed to operate with arbitrarily oriented imaging sensors and a procedure for epipolar transformation is required. The proposed solution of our consortium has ben to use a technique of reprojecting the image features onto a virtual plane, parallel to the plane of the three optical centers, and passing through a suitable position, at normalized distance from the optical centres [12]. In this way, with a very simple scheme and just 5 operations per image sample, it is possible to obtain the same conditions of an equivalent configuration with parallel optical axes. By choosing an appropriate reference system on the virtual plane one set of epipolar lines will become parallel to an axis, to simplify the matching process. Using this approach it is possible to reproject either the individual intensity values of the input images, or the edge samples belonging to the selected chains, or

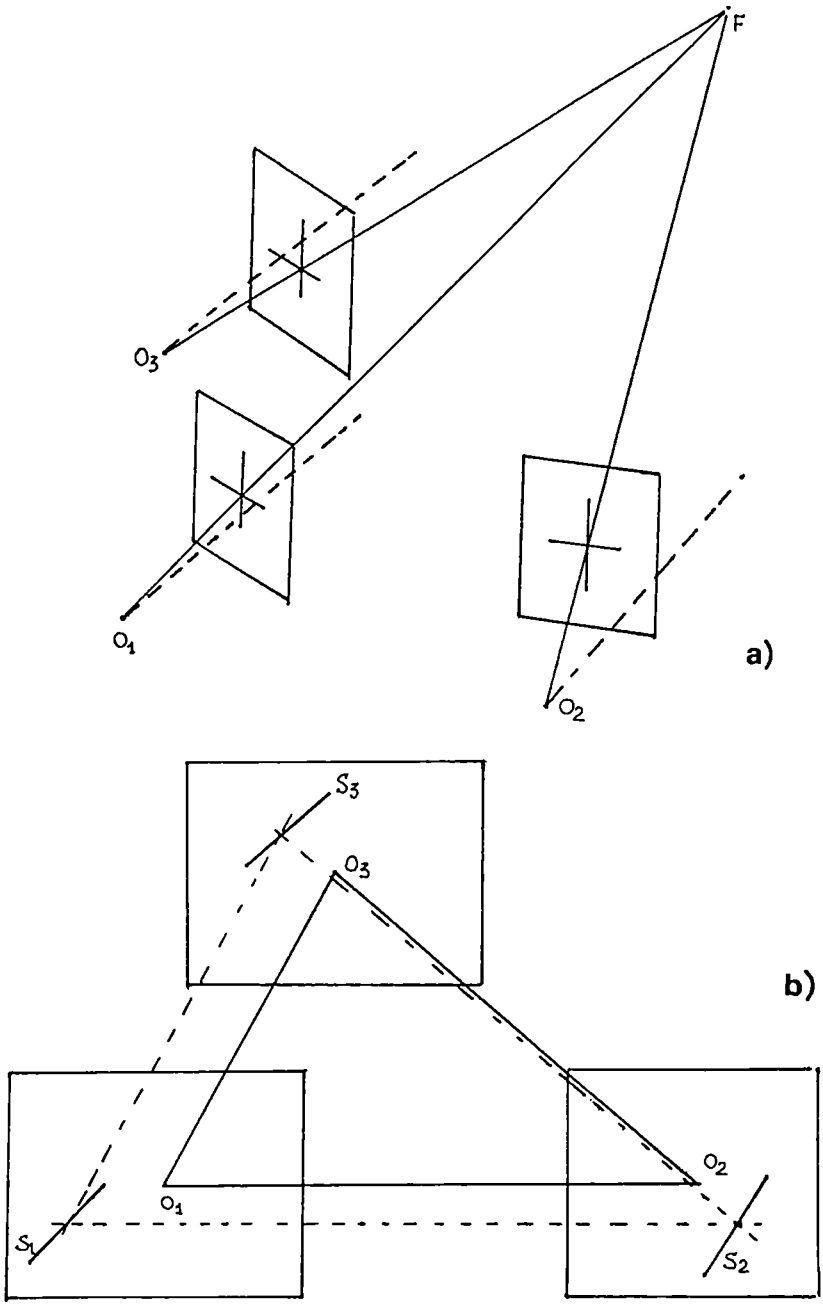


Fig.1 a) Geometry of trinocular stereovision; b) rectified images and normalized triangular geometry.

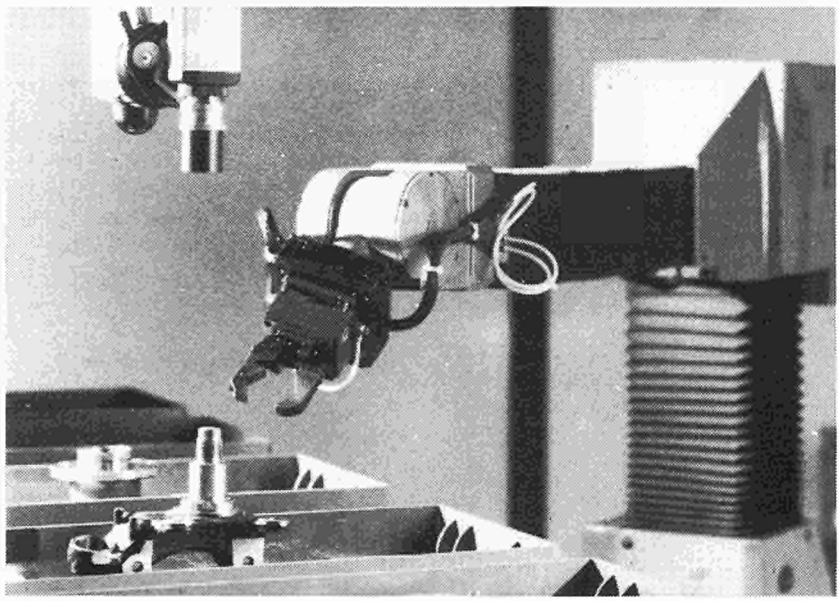
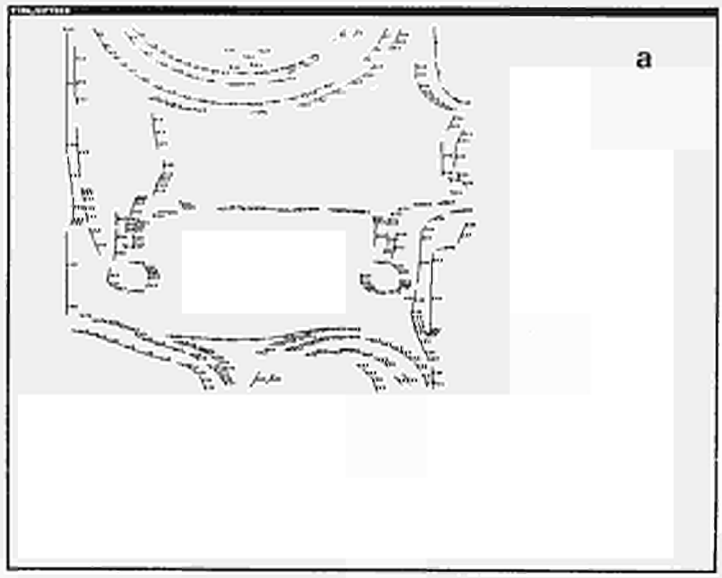
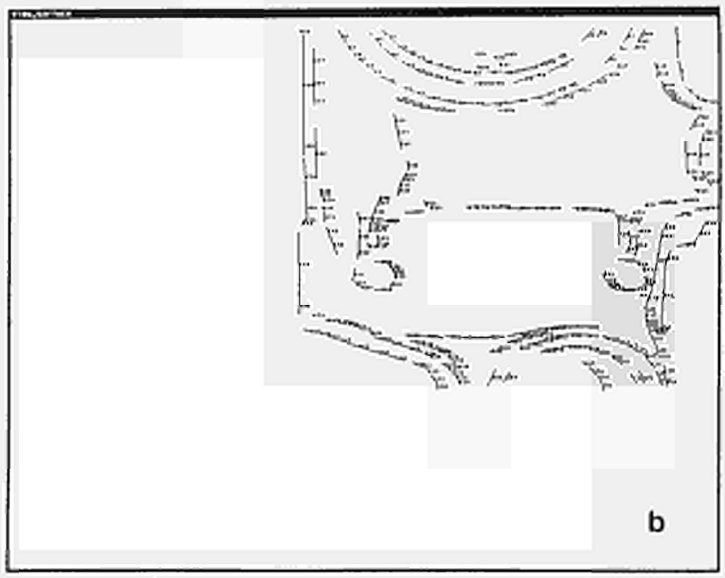


Fig.2 Robot arm manipulator used in our experiments.


a


b

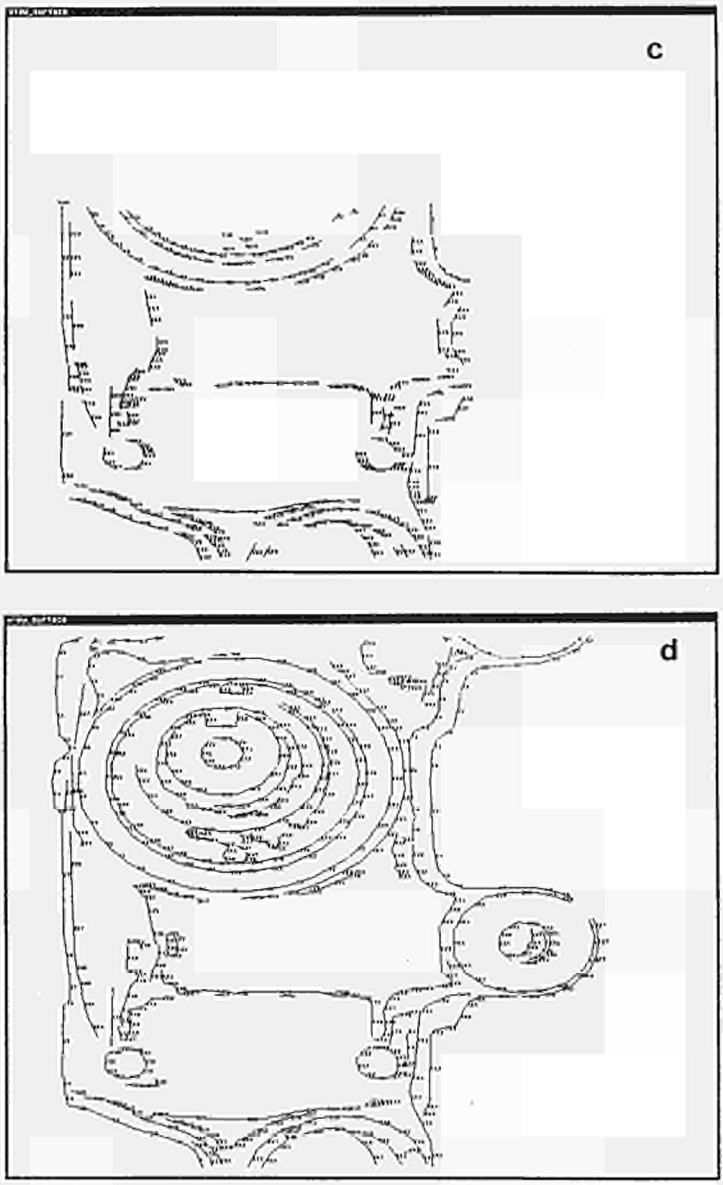


Fig.3 Stereo matching between the three cameras (a,b,c); simple example of camera translation where corrected matches are shown by putting the same number on the corresponding segments; only the common area which is visible by all cameras is displayed. d) result of polygonal approximation for one picture.

the endpoints of the linear segments, as required by our stereo matching procedure. In this way the number of operations involved turns out to be extremely low, without affecting the overall process.

6 ROBOT SYSTEM CONFIGURATION

Fig.2 shows the robot manipulator which will be used in our experiments. In the proposed experimental configuration one TV camera will be placed on the robot wrist and moved around the object to be described.

Three different views will be taken at different positions and the relative motion parameters used to calibrate the system within the precision of the robot movement. The optical calibration procedure which has been developed in our project can also be used to check for data consistency. Endpoints of the selected segments are rectified so that epipolar lines become horizontal in two of the images (as shown in fig.1).

A preliminar example is referred in fig.3 where a lot of circular structures were present. The purpose of this result is to demonstrate the correct correspondence, in the different views (a,b,c), of the selected segments which are labelled by a numeric code. It is worthwhile to observe that using a fine polygonal approximation it was possible to obtain, in this case, a reasonably good description of the curved contours in the scene.

Additional and more complete results will be referred at the Conference, for both environments of the mobile vehicle and the robot manipulator, to demonstrate the effectiveness of the results obtained in 3-D object reconstruction from passive stereo data within the Esprit project P940.

7 CONCLUSION AND FUTURE TREND OF RESEARCH

The natural development of the research on stereo vision is concerned with 3-D data representation starting from the obtained 3-D segments. The most promising solution consists in performing data interpolation using Delaunay triangulation. 3-D spline interpolation is also investigated to refine surface description. Another topic under investigation is the segmentation

of the scene, in order to detect homogeneous objects which are then described in terms of building surface primitives like planar surfaces, cones, cylinders, spheres. The subject of motion analysis is becoming more and more central to our project. Many algorithms have been already developed for 3-D structure estimation from motion parameters starting from point matches and line matches in the temporal sequence. A key problem under investigation is the development of an edge token tracker, to be able to track a linear edge feature during its temporal evolution. 3-D token tracker is also studied by matching 3-D line segments coming from stereo reconstruction. Major emphasis is now put in the cooperation of motion and stereo, where motion information is planned to be used to simplify stereo matching and viceversa 3-D stereo data may improve accuracy in motion estimation.

As a conclusion, the paper describes the major results and achievements of ESPRIT project P940, with particular attention to the stereovision approach. In particular we have adopted a trinocular system which makes use of linear segments as descriptive tokens in the scene. More detailed results of its performance in office scene reconstruction and industrial object representation will be presented at the Conference.

REFERENCES

[1] *Technical Annex ESPRIT Project P940, Depth and Motion Analysis*, Dec. 1985

[2] Canny J.,*Finding edges and lines in digital images*, MIT Artificial Intelligence lab., Cambridge, MA, Rep. AI-TR-720, June 1983.

[3] Deriche, R., *Optimal Edge detection using recursive filtering*, Proceedings of the First Int. Conference on Computer Vision, pp.501-505, June 1987, London.

[4] L.Borghesi, E.Giuliano, G.Musso, F.Cabiati, P.Ottonello, *Programmable modified systolic array for fast one- and two-dimensional convolutions*, J. Opt. Society of America, vol.3, N.9, 1986, pp.1561-1568.

[5] G.Giraudon, *An efficient Edge chaining algorithm* Internal Report INRIA, 1986.

[6] M.Berthod, *Polygonal Approximation of edge chains*, INRIA Internal Report, 1985

[7] N.Ayache, F.Lustman *Fast and reliable passive stereovision using three cameras* Int. Workshop on Industrial Applications of Machine Vision and Machine Intelligence, Tokyo, Feb. 1987.

[8] O.D.Faugeras, G.Toscani, *The calibration problem for stereo*, CVPR '86, June 1986, Miami.

[9] B.Caprile, V.Torre, *A TV camera calibration technique using the vanishing points*, Internal Report, Dept. of Physics, Genoa Un., May, 1987.

[10] *Comparison of Stereo Algorithms*, Report R 1.3.2., ESPRIT P940, June 1987.

[11] D.Ferrari, A.Pardo, *A 3-D contour description using range data*, Proceed. of the 2nd Int. Workshop on Time-varying Image Pr. and Moving Object Recogn., Firenze, 1986.

[12] B.Caprile, *Epipolar transformation by perspective projection*, Internal Report, Dept. of Physics, Genoa Un., Apr. 1987.

Project No. 393

DIALOGUES WITH LANGUAGE, GRAPHICS AND LOGIC

Ewan KLEIN

Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, Scotland.*

We describe work-in-progress on developing a system which allows the integration of natural language with graphics in knowledge base query and update. Achievements to date include the development of a logic-oriented semantic representation language for English, French and German, and the linking of deictic words (e.g *this, there*) to mouse hits. The system is implemented in Prolog, and consists of five major modules: Parsers, Dialogue Manager, Graphics System, Knowledge Base and Text Generator. In this paper, emphasis is placed on the natural language module, and we briefly discuss a proposal for the reduction of ambiguity in parsing prepositional adjuncts, in order to illustrate the utility of sorted logic in semantic representation. Prospects for adapting dialogue tableaux theory to theorem-proving, and for developing a syntax and semantics for graphics, are also outlined.

1. INTRODUCTION

You are looking at a video screen which displays a schematic map of Europe. Pointing at a node labelled *Stuttgart*, you ask "How much storage is available there?" The system responds with a bar chart showing the amount of free storage for refrigerated goods, liquids, and dry goods. Since you plan to unload 500 hectoliters of white wine in Stuttgart, you graphically manipulate the appropriate bar chart column so that it shows the reduced space available. As a result, the system's knowledge base is updated with a new value for liquid storage capacity at Stuttgart.

This kind of human-computer interaction is beyond current capabilities, but it is not unrealistic to imagine that we might approach it in the next few years. The ACORD project is intended to provide some of the fundamental conceptual tools which will enable the kind of mixed-medium dialogue illustrated above. In particular, ACORD is concerned with the task of integrating natural language and graphical representations at the level of common meaning structures. The problem is being attacked in the context of constructing a demonstrator where the update and interrogation of a knowledge base can be carried out in English, French or German in conjunction with interactive graphics; the domain of application is the logistics of

---

* This paper reports collaborative research by the following teams in ESPRIT Project 393 (ACORD): Laboratoires de Marcoussis; Section de Linguistique, Université de Clermont-Ferrand II; Centre for Cognitive Science, University of Edinburgh; EdCAAD, University of Edinburgh; Triumph-Adler AG; Fraunhofer Gesellschaft IAO; BULL; and Institut für Linguistik, Universität Stuttgart.

road transport. Logic plays a primary role in the project for at least two reasons: first, because the system is being implemented in the logic programming environment of Prolog, and second, because ideas from logic inform the theoretical underpinning of practically all the components. Finally, we note that one vital prerequisite for the easy integration of deictic gesture (i.e. pointing) into natural language dialogue is good speech recognition capacity, something which falls outside the remit of ACORD; we assume that the efforts of others will bear fruit in this area.

In this paper, we will give a general overview of the progress made to date. Like many other ESPRIT projects, ACORD involves the complex interaction of a large number of different organizations and individuals. The perspective offered here does not pretend to reflect this complexity in all its detail, but will be biased towards the natural language parsing components of the project.

2. DIALOGUE MANAGEMENT

The hub of the whole system is the Dialogue Manager (DM), since it controls communications between the other four subcomponents of the ACORD system, namely the Parsers, the Graphics System, the Knowledge Base, and the Text Generators.

To begin with, the DM mediates between the Parsers and the Graphics System, on the one hand, and the Knowledge Base on the other. This link allows the DM to channel Parser output and mouse events to the Knowledge Base, and also allows a certain amount of information to feed back the other way, since the Parsers have restricted access to Knowledge Base object-hierarchies and part-of relations in the course of resolving definite noun phrases. At present, only Prolog deduction is available to the DM, but further theorem-proving capabilities are under development, in order to aid with anaphora resolution and other disambiguation tasks. In addition, the DM is responsible for integrating graphics selection-events into the semantic output of the Parsers. This involves correlating deictic expressions such as *this, that, here,* and *there* with the identifiers of objects which have been referenced by mouse hits.

The other main logical connection concerns the flow of data between the DM, the Graphics System, and the Text Generators. The DM translates the results returned by the Knowledge Base into messages that can be conveyed to the generator devices for graphical and textual output. In addition, the DM has to decide on the kind of response that will be generated (graphics, text, or both), and on the amount of information that will be displayed to the user. In the current prototype, the DM only carries out 1-1 translations of Knowledge Base identifiers into Graphics System identifiers, and supervises the correspondence between the structure of objects displayed on the screen and the internal access paths to these data.

3. NATURAL LANGUAGE PARSING

There are sub-modules for each input language, namely German, French and English. The different teams involved in parsing have adopted somewhat different syntactic frameworks and parsing tools, though there is a common commitment to the general methodology of unification grammar (cf. Shieber [1]), according to which complex feature structures are a basic component of the formalism, with unification being the basic operation over such structures. Moreover, the task of parsing can be

viewed as a process of constraint satisfaction with respect to both syntactic and semantic requirements. Flexible parsing, in the sense of coping with agrammatical or otherwise corrupt input is not a goal of the project, but the treatment of anaphora and ellipsis is a major focus.

3.1. Semantic Representation

A notable feature of ACORD is that all the parsers deliver output in a common semantic representation language. Our starting point for semantic representation was the work of Kamp [2] on Discourse Representation Theory (DRT). Despite being translatable into first order logic, DRT offered a novel theory of quantifiers and pronouns which promised to allow a unified treatment of inter- and intra-sentential anaphora. We have now developed a sorted logic, InL, which retains Kamp's analysis of quantification and also incorporates a Davidsonian treatment of events.

A fundamental feature of InL is that each well-formed expression is of the form $[a]A$, where A is the body of the expression, and a is a distinguished variable, called the *index*. (1) illustrates the (simplified) InL formula for *A lorry is waiting*:

(1) [s][[x]lorry(x), wait(s, x)]

Here, s has the sort of states, and x has the sort of singular objects; conjunction between clauses is indicated by ','. Indices correspond to Kamp's [2] reference markers which have been entered into the universe of a discourse representation; except where they fall within the scope of universal quantification/implication, indices are implicitly subject to existential quantification. Thus, (1) says that there is an s and a lorry x such that s is the state of x waiting.

The index plays two important roles: since it is a sorted variable, it can provide a kind of typing information. Thus, we know that $[x]A$ is a nominal expression, and denotes a property of individuals, while $[s]A$ is verbal expression, denoting a property of states. (Of course, the sorting regime allows many other distinctions, say between singular and plural objects, or events and processes.) Moreover, the index marks an argument place that is open to further semantic specification, and thereby plays a role analogous to that of lambda-bound variables in conventional approaches to the compositional construction of semantic representations.

The parsers have a reasonably wide syntactic coverage, including constructions such as prepositional complements and adjuncts, obligatory control verbs, relative clauses, interrogatives, partitives and pseudo-partitives. As an illustration, let us briefly consider the analysis of prepositional adjuncts. Both the English word *in* and the French *à* are ambiguous (at least) between a locative and a temporal reading; thus, for example,

(2) Jean arrivera à Paris / à dix heures.
Jean will arrive in Paris / in ten hours

Consequently, we might suppose that the lexical entry for *à* contains a disjunctive list of semantic relations in the following manner:

(3) à, Prep, [spat_loc, temp_loc, ...]

At lexical lookup, the semantic representation of *à* will be added by a clause of the following form:

(4) adjunct([spat_loc, temp_loc, ...], x, y)

where *x* is the 'subject' of the preposition (in (2), the event introduced by the verb arrivera), and *y* is the object (i.e., either *Paris* or *dix heures*). In the case at hand, we can select the right reading at parse time by reference to the sort of the object argument. That is, suppose that *Paris* is entered in the lexicon as possessing a representation of sort SPATIAL, and suppose we have a general constraint of the form

(5) adjunct([spat_loc, ...], x, y) :- SPATIAL(y)

This allows us to derive the conclusion that a spatial reading of *à* is compatible with an NP such as *Paris*, but not with *dix heures*; in the best case, the disjunctive list of prepositional interpretations can be narrowed down to a single item.

3.2. English, French, German

Parsing of English and of French is being carried out within a version of categorial grammar which makes heavy use of techniques from unification grammar; this has been dubbed UCG [3]. As a grammatical framework, UCG departs from standard categorial grammar in two major respects. First, the grammatical object to the right of the categorial slash is not simply a syntactic category, but a complex of information (including semantics, syntax and phonology) which, following Pollard [4], we call a sign. Second, the use of unification allows us to build under-specified feature structures which in effect constitute polymorphic functions over the relevant grammar domains. Unbounded dependencies and pronominal anaphora are dealt with by gap-threading techniques (cf. [5], [6]).

The teams working on French grammar have devoted considerable attention to a variety of problem areas, including the syntax of clitics, auxiliaries, obligatory control verbs, interrogative constructions [7], and the semantics of ellipsis [8]. One novel step has been to extend UCG with a restricted form of function composition (cf. [9]) to handle *wh*-interrogatives and the interaction of clitics and auxiliaries. Parallel research is investigating the utility of using the attribute-value framework for the implementation of generalized phrase structure grammar, and of interfacing string grammar with DRT.

Parsing of German is being carried out within the framework of Lexical-Functional Grammar; thus each sentence receives a f(unctional) structure before being mapped into a semantic representation. It was decided to construct an extra level of syntactic representation in order, for example, to have the means to encode a variant of Chomsky's binding theory, to relieve the semantics of the burden of word order variations in German, and to get an appropriate starting point for the generation of different possible scope assignments to quantifiers and operators. Let us make the last point a little bit clearer. F-structures are acyclic graphs, i.e. their edges are not ordered with respect to each other. On the other hand, different scope assignments to quantifiers can be seen as the result of different orders of functional application. The algorithm (cf. [10], [11]) which we use for the mapping of f-structures to InL is based on this fact, i.e., it imposes different orders among the predicate and its arguments, thereby generating different readings without invoking different syntactic analyses or a device like Cooper-storage [12]. The fact that LFG, in contrast to the other linguistic theories employed, builds an extra level of representation allows for interesting comparisons between monostratal and multistratal analyses in the overall system.

Other topics of concern have been the analysis of non-finite constructions, separable verb prefixes, measure constructions, and the relatively free constituent order of German [13]. The current parser combines Tomita's generalisation of LR-parsing and the main ideas of the implementation described in [14].

4. THE KNOWLEDGE BASE

The KB has to support a number of functions. The foremost of these is to act as a repository for domain-specific knowledge, and to provide a mechanism by which user-supplied updates are incorporated into long-term storage. Another important function is to allow individuals to be incrementally defined by a series of partial specifications which are accumulated and propagated within the object hierarchy. As is now usual, frequently-used taxonomic and default reasoning is precomputed by classifying entities in an inheritance semi-lattice, based on the class/instance paradigm.

Objects and relations may be combined to form 'states-of-affairs' (which are either dispositions, events, or 'general'); these are encoded as straightforward Prolog facts [15]. The resulting inference system is a hybrid, in the sense that inheritance and resolution theorem-proving are both available; however, there is no need for an extra level of meta-inference control.

5. PROSPECTUS

A large part of future work will involve extensions to the functionality of the Dialogue Manager. For example, on the output side we wish to provide a capacity for over-answering, for the generation of references to graphical objects, for the use of elliptical expressions or even single word responses, and for the selection of the appropriate graphical representation.

5.1. Deduction

As a further aid to resolution, we have been developing a deductive capacity for the Dialogue Manager [16]. Since Kamp's Discourse Representation Structures are logically similar to semantic tableaux, we have focussed on tableaux methods for theorem proving, particularly dialogue tableaux theory. This views the logical deduction of a formula ϕ from a database Γ as a debate involving two idealized dialogue partners. The Opponent, having conceded Γ, attacks ϕ; the deduction succeeds if the Proponent manages to defend ϕ against these attacks. Dialogue tableaux theory is preferable to conventional semantic tableaux in that it allows the characterization of more logical calculi (intuitionistic and minimal, as well as classical logic), and is more readily implemented.

5.2. Graphics

The bulk of work on graphics that has been implemented so far is a Prolog/GKS binding [17]. A primary goal of this work has been to reconcile the declarative style of Prolog with the inherent procedurality of GKS.

Research is now underway in attempting to determine the extent to which drawings can be analyzed like expressions of a language, possessing both a syntax and

semantics. Given a set of syntactic rules which are used to generate or interpret some range of drawing objects, it should be possible to arrive at a domain-independent semantics for the relationships between the constituents of the drawings, and to operate on the meanings in a way that is familiar from syntax-driven algebraic semantics. However, the syntactic structures of drawings are not directly relevant to their domain-specific meanings, since they are subject to a variety of semi-conventional interpretations which are related in only an unsystematic manner to their spatial-representative aspect. We are aiming to identify the basic syntactic constituents of drawings by reference to the communicative context, rather than by reference to formal structure alone. This context is often significantly determined by explicit textual annotations of drawings, but also depends on the history of the ongoing dialogue between user and system.

ACKNOWLEDGEMENTS

Although Klein is nominally the author of this paper, it represents contributions from many participants in the ACORD project. Thanks are particularly due to Jo Calder, Annick Corluy, Werner Frey, Gerhard Heyer, Jaap Hoepelman, John Lee, Mark Moens, Celestin Sedogbo, and Henk Zeevat.

REFERENCES

[1] Shieber, S. M., An Introduction to Unification-based Approaches to Grammar (University of Chicago Press, Chicago, Illinois, 1986).

[2] Kamp, H., A Theory of Truth and Semantic Representation, in: Groenendijk, J. A. G., Janssen, T. M. V. and Stokhof, M. B. J., (eds.) Formal Methods in the Study of Language, Volume 136, (Mathematical Centre Tracts, Amsterdam, 1981) pp. 277-322.

[3] Zeevat, H., Klein, E. and Calder, J., Unification Categorial Grammar, in: Haddock, N. J., Klein, E. and Morrill, G., (eds.) Edinburgh Working Papers in Cognitive Science, Volume 1: Categorial Grammar, Unification Grammar, and Parsing (Centre for Cognitive Science, University of Edinburgh, 1986), pp. 193-222.

[4] Pollard, C. J., Lectures on HPSG, unpublished MS. (CSLI, Stanford University, 1985).

[5] Pereira, F. C. N., Extraposition Grammars, American Journal of Computational Linguistics 7 (1981), pp. 243-256.

[6] Johnson, M. and Klein, E., Discourse, Anaphora and Parsing, in: COLING86, Institut für Kommunikationsforschung und Phonetik, Bonn University (1986) pp. 669-675

[7] Baschung, K., Bes, G. G., Corluy, A. and Guillotin, T. Auxiliaries and Clitics in French UCG Grammar, in: Proceedings of the Third Conference of the European Chapter of the Assocation for Computational Linguistics, Copenhagen (1987).

[8] Sedogbo, C., Extending the Expressive Capacity of the Semantic Component of the Opera System, in: COLING86, Institut für Kommunikationsforschung und Phonetik, Bonn University (1986) pp. 23-28.

[9] Steedman, M., Dependency and Coordination in the Grammar of Dutch and English, Language 61 (1985) pp. 523-568.

[10] Reyle, U., Grammatical Functions, Discourse Referents and Quantification, in: Proceedings of the Ninth International Joint Conference on Artificial Intelligence, University of California at Los Angeles (1985), pp. 829-831.

[11] Frey, W. Syntax and Semantics of Some Noun Phrases, in: Laubsch, J., (ed.) Proceedings of GWAI 1984 (1985).
[12] Cooper, R., Quantification and Syntactic Theory (D.Reidel, Dordrecht, 1983).
[13] Netter, K., Getting Things Out Of Order: An LFG Proposal for the Treatment of German Word Order, in: COLING86, Institut für Kommunikationsforschung und Phonetik, Bonn University (1986) pp. 494-496.
[14] Eisele, A. and Dorre, J. A Lexical Functional Grammar System in Prolog, in: COLING86, Institut für Kommunikationsforschung und Phonetik, Bonn University (1986) pp. 551-553.
[15] Heyer, G. PROLOG for Processing Natural Language Semantics.and Data Bases, ESPRIT Conference on Databases (Venice, 1986).
[16] Hoepelman, J. and Machate, J. Dialogue Theory, Theorem Proving, Database Questioning and Natural Language, in: Katz, P. (ed.) ESPRIT '85 (North-Holland, Amsterdam, 1986).
[17] Krishnamurti, R. and Sykes, P., A Graphics Interface to Prolog, in: Katz, P. (ed.) ESPRIT '85 (North-Holland, Amsterdam, 1986).

Project No. 311

A D K M S: Advanced Data and Knowledge Management System

Juergen Peters

Nixdorf Computer AG, EDBS 42
Berliner Str. 95
D-8000 Muenchen 40

Abstract

The main goal of ESPRIT project 311 is the design and the prototypical development of a system which manages very large databases and knowledgebases in an integrated way.

The system under development has the following main modular components:

- a rule-based Natural Language Handler for convenient and flexible access using different knowledge sources (lexical, grammatical, domain, application, world knowledge),

- the BACK system (Berlin Advanced Computational Knowledge representation), a further development of KL-ONE, with restricted, but efficient inference mechanisms,

- a Knowledge to Data Transformation System for generating automatically the Logical Schema of a Relational Database Management System (RDBMS) from the knowledge representation of the BACK system,

- a RDBMS with an extended SQL as its data manipulation language which can compute certain recursive queries (e.g. the transitive closure) more efficiently than a "traditional" Inference Machine when the amount of data is very large and when the data first has to be fetched from secondary memory.

The integration of these components into an advanced data and knowledge management SYSTEM will be shown formally by presenting the used principles of software engineering, and in substance, by presenting the main system components and their contributions in creating the data and knowledge bases and in evaluating a natural language query using the inferential capabilities of the system.

Introduction

Since December 1984 three major European computer manufacturers, namely Nixdorf Computer (prime contractor), Olivetti, and Bull, and computer science departments of four European universities, namely, TU Berlin (partner) and Bologna Turin, and Hildesheim (subcontractors) are involved in ESPRIT project 311.

In the design and realization of the system different approaches are used for the different components. In particular, a fruitful cooperation has been started between researchers and system developers educated in the traditions of Artificial Intelligence, Database Engineering, Logic Programming, and Software Engineering.

With such a hybrid approach a clean system design was necessary to be able to integrate successfully the components into the ADKMS. As we shall outline in the following a new paradigm of system development, the corner- to-corner-approach, had to be invented to cope with the problems of system integration of such different components.

Overall idea of the system

We consider the following parts of the system's idea to be the central part of the project which we are about to realize:

1. Design and implementation of a knowledge representation formalism which takes the conceptual/terminological structure of the domain as the structuring principle of the database scheme to be developed, which is able to inherit attributes of concepts and therefore is suited to understand natural language queries, which is able to represent knowledge and draw inferences upon the stored data, which supplies a mechanism to input data into a database, and which supplies a query language for the stored data.

2. Design and implementation of natural language interface capabilities able to translate a natural language query into an unambiguous semantic query and transfer the latter one into a query of the knowledge representation formalism.

3. Design and implementation of an extended relational database which has acceptable access time with very large amounts of data.

4. Design and implementation of algorithms which automatically transfer the knowledge representation into the extended relational database query language.

Design Principles

The general objective of the project concerns the design of a complex data and knowledge management system that can be inquired through natural language. The project presents both knowledge representation and knowledge management problems. The former problem deals with the kind of semantics that is made available to the natural language handler for solving interpretation ambiguities. They are the same semantics that can be used to answer user queries and to constitute the dictionary of the underlying database of instances. The latter problem deals with optimal knowledge and data organization into a relational database schema taking into account the DBMS features and having in mind expectations on the use of the knowledge base.

The two kinds of problems refer to different research fields and their solution calls for different cultural backgrounds. In other words, a good approach in the design of the overall system seems to be an initial separation of the studies in the two fields, and a recursive integration of proposals and results to obtain a complete, homogeneous system.

This can be done provided that an intermediate interface is established between the two parts in the early phase of the project. The interface must be the real cornerstone of the project, so as to represent, at the same time, a formalization of the normal language handler output and a significant input to the KB design process. Furthermore, its definition must be already available at the beginning of the project (at least, to a sufficient extent) to avoid delays in performing the tasks, which mainly depend on it.

The KL-ONE representation formalism has been identified as an interface that satisfies the above needs. Its management, the introduction of possible extensions to meet the project requirements, as well as the performance of inference operations, have been considered activities of one of the project's tasks.

Natural language handling now becomes an autonomous research activity, with the only constraint to express KB requests in terms of expressions of the knowledge representation formalism. The choice of developing three separated prototypes for the three languages of the project partners is discussed elsewhere #. Here we can only notice that all of them are built having available a language for issuing requests against the KBMS.

On the other side, the KB access requests constitute inputs to the KBMS. They must be translated into sequences of DB operations to be performed by the DB management system which stores concepts and instances. In managing long files of instances no real new problem must be faced with respect to those examined in the database literature. Instead, many problems arise, when concept files are accessed to answer user queries. In particular, the problem of performing frequently transitive closures on concept fillers has been identified as fundamental at the very beginning of the project. In few words, it has been recognized the necessity to introduce a second interface to separate material DB management from conceptual-to-relational KB mapping.

An augmented relational database language, XSQL, which includes operations for transitive closure and other navigations along direct acyclic graphs has been adopted as the second corner interface of the project. Thus, two last tasks remain to be performed:

- bridge the gap between knowledge representation and selectional schema formalism, and

- design an efficient augmented DBMS to be used in storing and managing a knowledge base.

In conclusion, the Advanced Knowledge and Database Management System which is mentioned in the title of this project is not designed as a whole, but as the integration of modules, each covering one segment of the path that goes from the natural language to the elementary access operators to the secondary storage. Then a unique design methodology cannot be conveniently adopted for all the modules. It has been preferred an approach that establishes two corner interfaces, thus dividing a multi-disciplinary problem into single- field sub-problems. Now the project consists in covering the space that divides one corner from the other and the extreme interfaces (natural and relational) from the near corners.

Layered Architecture

A data flow analysis of this architecture shows that the different modules of the system cope with different kinds of data; every module handles data with a different semantic content. This observation shows that we can think about the systems in terms of abstract layers, where every layer connects two different levels of expressiveness in the processed statement. Such layers are shown in table 1.

This kind of analysis can help to understand how the system is organized, where the knowledge about the various component resides and how it is used.

We can see the system as a set of pipelined modules. The data flow starts from the NLH and goes through all the modules to the extended data base. However it is not a one-to-one query translation process: every module elaborates the incoming query in a complex way, and more than one query are generated for the below level. So the query processing is a one-to-many translation, performed at the various levels.

The system is able to answer to a query with inherently complex semantics. This is achieved through a number of analyses on the incoming query; every step translates the query in a less expressive language, i.e. in a language that handles objects simpler than the objects of the previous language.

The first form of the query is the natural language: the NL query represents a question made to the knowledge base by a user.

This query is analyzed by the appropriate NL handler; in this process the syntax tree of the query is obtained, and a semantic analysis is performed, obtaining a representation of the semantics of the sentence in terms of the BACK formalism.

Note that the transformation Natural langage -> Syntax ->Semantics is performed inside every NL module, and every module uses its own strategy, so the two step schema is only an abstraction: in the actual modules the two steps are intermixed, and possibly other intermediate representations are used.

The semantic query (represented in terms of the BACK Tell/Ask-Interface) is then analyzed from the BACK module: depending on the query this task may require a considerable amount of deduction. These deductions are implemented in terms of operations on a complex data structure built using the Object-Oriented primitives provided from the knowledge management level.

These operations are actually implemented by the run time support transaltering them in extended relational database operations, performed by the data base module. So from the expressive power point of view the O-O oriented layer* provides a more network oriented implementation language for BACK.

In the following table 1 the different levels of abstraction are outlined:

Representation	Abstraction layer
Natural language	Human knowledge.
NLH internal	Syntactic and linguistic dependent semantics.
BACK Tell-Ask-Query	Domain dependent semantics.
O-O operations*	Data structure semantics.
Ext. Relational DB	Relational algebra semantics.

Table 1: ADKMS Abstraction Layers

One important point to stress is that not all transformations are done at run time; i.e. not all the modules are interpreters: in particular the O-O operation* called by the BACK system will be compiled directly in database operations; only data format conversion will be performed at run time. Compilative techniques may be applied in other modules, too.

Various knowledge sources are needed to process completely a query.

These sources may be used at run time or compile time, depending on the modules. The following table 2 sketches the various knowledge sources used, and their approximate size:

From	To	What
NL	Syntax tree	Syntax rules (70)
Syntax tree	Ling. dep. knowledge	Semantic rules(70)
Surface Know.	Deep Knowledge	Transl. rules (-)
Deep Knowledge	O-O query*	BACK metaschema, domain model(200)
O-O query*	Ext. relational queries	ABox-,TBox- concept schema, logical schema. (200)

Table 2: ADKMS Knowledge Sources

To these should be added the heuristic knowledge needed by the compiler on BACK and on the domain modelled.

Load Balancing and Computational Power

It was observed before that the queries are Knowledge base queries and not data base queries: this implies that some deduction activities are involved in the query answering process: This consideration is important when the load balancing between modules is considered: we consider here two different kinds of "load balancing", given one from a pragmatic point of view, the machine-time load balancing, and the theoretical one, the computational power distribution between modules.

Of course we consider here only the actual "inference engine" of the KBMS, and not the NL modules. There seem to be two contradictory goals in the project (from this point of view): first we want to use the data base as a work horse, in the sense that we want the data base to perform most of the computation of the system to be able to handle very large amount of data. Secondly we want to obtain the expressive power of a knowledge representation system over the stored data.

The result is a compromise between these different goals: increasing the computational power of the data base to the transitive closure power, and a careful implementation both of the TBox and the ABox over the data base.

In particular the mapping to the data base is performed in such a way that all the informations on instances are stored as in a traditional db application so that the full power of a data base is available for queries on instances.

However, it seems that the computational power of the extended data base is not sufficient to implement all the BACK deductions. So a part of the computational power is to be implemented in BACK, using the capabilities of Prolog. Therefore the job of the knowledge management level is to map part of the BACK computation on a data base, with the main goal that all the computations performed on large amounts of data (usually computation on instances) are actually performed by the data base.

Integration Planning

In the following we show which efforts have been spent to ensure that integration can take place successfully:

- the organization of the system is modular
- the functional layer structure, and
- the functional dependency structure,

give the framework to avoid the duplication of labour at different levels and components.

So, it is assured that one task is done only once at a well defined place. The integration is possible because of the strategy chosen and described above. It is also possible at the formal implementation level, because as one can see from the following picture that all the "upper" components are implemented in a common dialect of PROLOG (P311-Core-Prolog) which is an extension of the PROLOG described in Clocksin / Mellish. This makes ADKMS independent of specific operating and computer systems and allows to port the programs of the different project sites and to interface the modules.

Functionality Dependency of the Components of P311

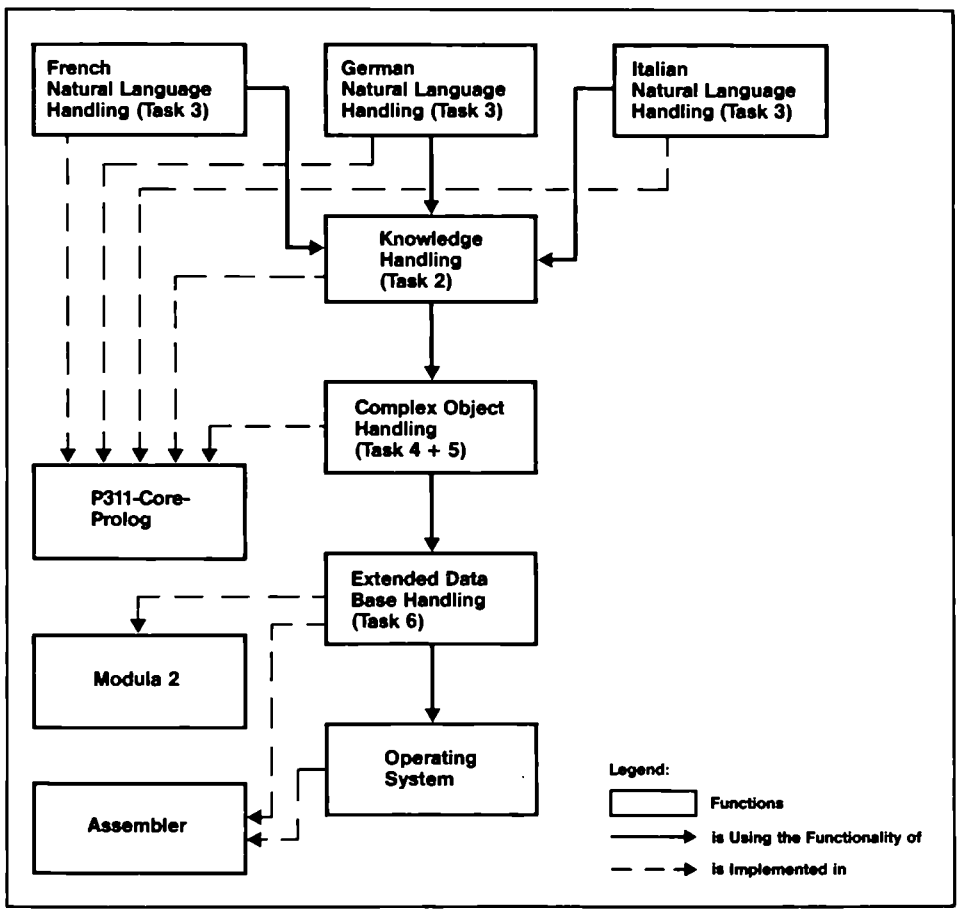


Figure 1: ADKMS Architecture with Functional Dependencies

The extensions of the DBMS (on the top: the interface language, Extended SQL, within the DBMS), the new data structures (DAG) and operations on them are implemented in Modula-2 in the first prototype, such that the integration with the upper level has to be done under the operating system. Fortunately, there exist in all standard operating systems (UNIX, VMS, VM/CMS, MS/DOS) conventions to combine runtime systems of different programming languages.

The German Natural Language Handler

The NLH takes a typed-in natural language expression (e.g. a German sentence) as input, analyzes it syntactically, transforms it into a wellformed expression of the Semantic Representation Language SR, controls the interaction of different knowledge resources and reduces the final SR-expression to a BACK-formula.

The input routine accepts all German conventions of orthography (e.g. lower and upper cases) and transforms the input string into a list of words tractable for PROLOG-clauses.

The analyzer is based on Chomsky's "Barriers"-model and driven by a depth-first left-corner-parser. It builds up a syntactic structure according to X-bar-theory as given in "Barriers" and takes case-marking cross-read with theta-expectations as main clues for the indication of thematic relations.

Single words (or idiomatic expressions) are mapped either on logical operators (and, or, not a.s.f.), on BACK-relations/-concepts or on procedural commands ("print", "sort", "calculate" and the like). Depending on the result of the syntactic analysis a proper SR-expression is produced.

SR is based on first order predicate logic (PL1) with a kind of enlarged vocabulary to allow for calling procedures and/or single system components as well as for doing arithmetics (non-PL1 quantifiers).
There are similarities to SR like the ones used by LOKI or PALABRE.

Certain lexical ambiguities, for instance, are resolved on this level by checking the context against interpretation rules. Those procedures are controlled by an annex of the SR, the SSV (Semantic SuperVisor).

Finally the result of all SR-processes is changed into a BACK-formula (by the socalled Knowledge Interpreter KI) and handed over to the BACK system.

The Italian Natural Language Handler

The parser of the Italian NLH is based on the production rule formalism. It is mainly deterministic, although in some cases the backtracking mechanism is still used. The rules that embody the linguistic knowledge (syntax) aim at building a dependency tree representing the relationships existing between the words appearing in the input sentence.

The dependency tree is composed of nodes of different types, the most important of which are REL (RELation, mostly containing verbs), REF (REFerent, containing nouns and pronouns), CONN (CONNector, for prepositions; a special filler - UNMarked - is used to indicate that the CONN node connects an unmarked case to its verb), ADJ (ADJectives) and DET (DETerminers). When an input word is analyzed, its category is retrieved from the lexicon (of course, the word can be ambiguous - i.e. it may belong to different categories; the NLH is able to handle this situation, but we will not consider it here for the sake of simplicity). On the basis of the category, a packet of rules is triggered; the preconditions of the rules belonging to the packet are tested; they involve elementary predicates that inspect the current situation of the dependency tree and possibly require a lookahead (two-word maximum) on the following piece of sentence. The action part of the rules involves the creation of new node instances, their attachment to a specific node of the existing dependency tree and the insertion of data into the nodes.
For instance, in a query starting with the word "Which", the analysis of this word would make the system verify (by means of a lookahead) whether it is a pronoun (as in "Which is ...") or a question adjective (as in "Which employees ..."). In the first case, it is assumed to be a case of the main verb, so that a new CONN node is created, it is attached to the main REL node (existing from the very beginning) and it is filled with the keyword UNM (see above); then a new REF node is built and it is filled with the input word. Then the next input word is scanned and the process continues in similar way.

The semantic processing is synchronized with the syntactic one. As soon as two nodes containing content words (this is not the case of the "Which" example discussed above) are attached, either directly or by means of a CONN node, the semantic interpreter is triggered. It verifies that the connection is semantically acceptable and, in the affirmative case, it returns some bindings that express the word senses compatible with the phrase (more precisely, a binding for the upper word and a binding for the lower one). These bindings will consist (when the NLH will be fully operational, i.e. in version 2) in nodes of the BACK net, thus specifying which are the BACK concepts referred to in the input expression.

In case no binding is possible, the connection which the check referred to is not acceptable; in this case the system attempts to restructure the tree in order to produce an alternative interpretation. This is done by means of "Natural Changes", a set of specialized rules we will not describe here, which have a function similar to the one of "well-formed substring tables" in more standard parsers. If the Natural Changes do not succeed, then the system resorts to backtracking to try to restart the analysis from a previous choice point.

A Short Overview of the BACK System

This section gives a short overview of the BACK (Berlin Advanced Computational Knowledge representation) system. The BACK System is a hybrid presentation system. It has some similarities to well-known hybrid systems like KRYPTON and KL-TWO. It also consists of two main formalisms, one for representing the terminological knowledge of a domain, the TBox, and one for representing the assertional knowledge, the ABox. The TBox is further development of KL-ONE with emphasis on computational tractability of the selected and implemented language constructs, the epistemological primitives. The ABox is a development of TU Berlin which incorporates the possibility of representing incomplete knowledge in a limited but tractable manner.

The main emphasis in developing BACK is on the balance of the TBox and the ABox language constructs. A balanced hybrid system differs from competing approaches based on the 'the more the better'-philosophy in this way, that each construct proposed as a language construct should play its role in drawing inferences and ensuring consistency by tractable algorithms, designed for realistic amounts of knowledge.

This direction of development resulted in a first prototype implementation of the BACK system with major differences to similar approaches like KRYPTON or KL-TWO as well as to e.g. OMEGA. This implementation was undertaken with a selected subset of PROLOG (KIT CORE Prolog) for ensuring portability of the BACK system and it is running at present on IBM with Waterloo Core Prolog and M-Prolog, on Symbolics with LM Prolog, and on Nixdorf Targon with M-Prolog and Ifprolog. It was evaluated with some domain models with promising results.

The further development of the BACK system is directed in enlargening the inferential capabilities of the system by techniques for which tractable algorithms are known. This will result in a further inspection and refinement of the language and further developments of the underlying inference engine.

One field of research in this direction will be the analysis of control- and meta-knowledge as a source for direction of lines of inferences to ensure efficiency of the inference engine in coping with realistic amounts of knowledge.

Connection BACK - Data Base Management System

The task consists of realizing a mapping between the BACK-System and a Relational DBMS. An approach based on a structural mapping from the primitives of the BACK language into DB relations (similar to the one investigated with the KEE-System) is under development and will serve as a starting point for further investigation on extending this line by a partial conceptual mapping between BACK and the DB.

The division of the work into subtasks is the following :

- Structural mapping from BACK language primitives into DB relations

- Mapping the TBox taking into account the classifying task

- Mapping the ABox without the contexts taking into account the constraint propagation technique

- Selecting the appropriate structure and optimization technique and achieving the mapping of the ABox partitioning (contexts)

Conceptual mapping between BACK and RDBMS

General objective of this subtask is to achieve system efficiency by mapping high-level knowledge representation into KB relational representation. This is obtained in two steps:

1. Design: the design of the KB relational schema for the TBox and ABox component are obtained starting from a formal description of the knowledge representation formalism (BACK) and take into account application dependent information (the TBox content and the TBox and ABox usages).

2. Run-time support: user queries issued against the KB are translated into XSQL operations. The translation is referred to the KB schemata produced in the previous step and takes into account the effects of the addition of new concepts to the TBox, due to the classification and realization mechanisms of BACK.

We can identify four main goals to be reached within this task:

- Definition of a design methodology to derive the TBox logical schema from a formal description of the TBox structure, contents and from the expected operations to be performed.

- Definition of a design methodology to derive the ABox logical schema from the contents of the TBox and taking into account the ABox semantics and operations.

- Design of the Run-time support module which translates the BACK operations into XSQL operations.

- Implementation of a Run-time support prototype, to be integrated with the BACK system and the ORACLE relational DBMS.

TBox relational design

The design methodology proposed has been developed taking advantage of previous experience on design methodologies for database applications and is applicable to knowledge representation models based on KL-ONE, as it is the case of the BACK system.

The methodology consists of two major phases:

(1) conceptual design and (2) logical design.

In phase 1 the designer has, as input, a description of the meta concepts of the knowledge representation model adopted. The meta concepts are expressed by means of a formalism based on an extension of the Entity-Relationship model. This extension has been developed to allow more flexibility in the knowledge organization with respect to the standard E-R models used in the database design environment. The result of the first phase is a conceptual schema. This kind of output is particularly suitable for the second phase of the methodology, which generates the logical schema.

In phase 2 the logical relational schema is generated, starting from the conceptual schema. To this purpose, a set of rules has been identified. The rules are designed to operate with two types of input:

a) an EER conceptual schema of the meta-concepts;

b) a qualitative description of the most frequent operations on the schema and quantitative information on the use of the above operations.

On the basis of this types, we have two levels of input:

level 1 - only type a) input
level 2 - both type a) and type b) input.

For each input level, a relational schema can be derived. Level 1 produces a very general schema, based only on application domain independent informations, where the way of using the knowledge base is not considered. Level 2 is available only when the application domain and its usage has been defined. This latter level generates a relational schema optimized with respect to the number of logical accesses to the relations storing the knowledge base. System performance can be further improved by a physical design step, where the choice of the file structures and the access methods to the stored information is performed.

ABox relational design

Our aim is to design a ABox relational schema which guarantees efficient knowledge base interaction.

The structural mapping approach proposes a storing of the ABox consisting in a small number of standard relations whose structure is independent from the contents of the TBox (i.e. a relation to store all the role links between concept instances, another relation for all the concepts instances and so on).

Usually, an interaction with the knowledge base, especially for the assertional component, involves only a small part of the stored knowledge (i.e. retrieving the instances linked to a given concept instances by a given role). For this reason it would be useful to have a partitioning of the knowledge, so as to limit the search space for each retrieval operation.

This requirement becomes crucial when facing with very large knowledge / data bases. In fact, the relations generated by the structural mapping are very large and the access to a small part of the large relation can be very expensive (i.e. the retrieval of the instances linked to a given concept instance by a given role requires the sequential scan of all the role links).

A first possible solution might be the adoption of indexing techniques, as it is usual in the DB environment (i.e. we could build an index on the pair <concept-type-identifier, role-name> to address all the instances linked to each instance of a given concept by a given role). This approach would improve system efficiency, for the retrieval operations, even if it generates some additional costs <index structures require additional storing space and maintainance costs in case of update operations).

For an alternative solution, we can observe that TBox contents define a natural way of partitioning the ABox. In fact, it suggests to generate one or more relations to store the instances of each TBox concept. In the literature we find few, recent proposals on this guideline and we feel it is very promising for our project, since:

- it provides an effective partitioning criterion based on the application domain semantics and can improve system efficiency;

- the resulting structure is flexible enough to support dynamic enrichment of the TBox contents (as done by BACK during the classification operation), since the creation of new relations does not present real problems.

This solution can give raise to some complications due the very high number of relations generated when the TBox contents increase. For this reason we shall investigate an improved solution which reduces the number of relations. In the resulting schema the concepts which are close in the concept hierarchy are, if possible, clustered in the same relation, on the basis of their common roles.

The conceptual mapping of the ABox into a relational schema will be generated by using an extension of the design methodology which has been developed for the generation of the TBox relation schema. This is a kind of compilation process which considers the TBox contents as input information. Moreover, the compilation process generates a mapping table recording the correspondences between the TBox items and the relations where their instances are stored.

The compilation approach appears feasible, since the application domain structure is supposed to be completely described and stored in the TBox before the ABox relations are filled. It is task of the run-time support to deal with the subsequent extensions of the TBox determined by the classification activity.

Run-time support design

The aim of this subtask is to define a communication language between the BACK module and the DB and to translate the operations on data issued at run-time by BACK into extended relational operations. The guidelines are the following :

- classify the operations on data issued by BACK and single out precisely their semantics.

- allow the dynamic extension of the TBox when the new concepts are added.

Prototype for the run-time support

The realization of the prototype is necesssary to verify approaches and techniques developed in the previous subtasks. Moreover, integration of the run-time support prototype with those concerning the BACK system and the underlying augmented relationa DBMS allows demonstration of the feasibility of a KBMS for large amounts of knowledge.

Expecially the BACK system has incorporated mainly two inferential capabilities:

- For the TBox inferences the classifier process serves as the implementation of the taxonomic inferences .

- For the ABox inferencies the constraint propagation process serves as the implementation of the ABox completion inferences.

According to the demands of the conceptual mapping process these two implementations of basic inferences has to be adopted.

Database and Database Query Language

Common relational query languages like SQL lack some of the expressive power required for knowledge base applications.
In particular, we argue that a substantial and frequently used part of the inferential power should be directly supplied by the underlying database system.
Analyzing a collection of examples we intend to support various types of (normally recursively expressed) queries on graph relations. Thus we defined a graph manipulation kernel as an extension of SQL. This extended language, called XSQL, has features to generate a basic set of paths (in a directed graph), to select the relevant paths from it by some conditions (including those referring again to a pathset) and to produce the desired output. We proposed an implementation of XSQL based on careful adaptions of graph algorithms, im particular of algorithms for computing the transitive closure. Supporting a huge amount of data the algorithms are to be tuned to minimize access to secondary storage.

Internal Operators for XSQL

We shall define a state diagram the nodes of which represent internal structures of database relations and the edges of which represent algorithms (internal operators) marked with cost estimates.

Runtime simulation XSQL and Demo Preparation

In order to rapidly demonstrate the expressive power of XSQL and its usefulness within ADKMS the XSQL language will be simulated be means of a common relational database system, ORACLE. For this purpose a subset of the proposed XSQL will be translated into procedures that call SQL routines. The simulation should provide further experiences with the language and enable a demonstration of its role within ADKMS. Furthermore, this simulation will then be used for the connection of the BACK system with relational DBMS.

Implementation of XSQL

XSQL and some parts of SQL will be implemented using the internal operators as they are proposed in previous deliverables Some preliminary studies on calculation along paths and on optimization will also be attempted.

The final implementation should demonstrate the feasibility of our approach to move inferential power to the underlying data base system. It will also indicate how to incorporate the implementation into common relational data dase systems.

Conclusion

The impact of the research in ESPRIT project 311 is very great because there is an urgent need

- to enhance DBMS with greater functionality without loosing efficiency ("intelligent database systems"),
- to make feasible the use of sophisticated knowledge based system in case when large amounts of data and knowledge are involved ("expert database systems"),
- to make available not restricted natural language handlers which can cope with the large variety of linguistic and dialogue handling problems, which are transportable, that is, not tailored for one specific domain.

The chances that our project will be able to meet the requirements of the user are good because we base our research and development on solid grounds of already accepted standards in A.I. and database research and engineering.

At present the project may be ahead of the industrial schedule of commercially exploiting systems like ours, but, firstly, there are still many tasks to be done until the ADKMS is ready, and, secondly, the industrial schedule may well speeden up because of users' demands. It goes without saying that even if we me meet all our requirements there are still many efforts to be spent until something like a product will be available.

---

see deliverable D4 of P311 (available to other ESPRIT projects) The work on the French Natural Language Handler was finished within ESPRIT after delivery of a first prototype. It is not described in this paper.

* The plan of compiling knowledge and to use compiled knowledge at run time is no more included in the new workplan of the project valid for the time span until May 1988.

Acknowledgement

The project is partially funded by the ESPRIT Programme of the C.E.C. (Advanced Information Processing, P311).

Project No. 96

The Expert System Builder (ESB)

Finn R. Jensen, Søren T. Lyngsø A/S
Lyngsø Alle,
DK 2970 Hørsholm, Denmark
Tel: + 45 2 572500 ext 4365.

Alessandro Dionisi Vici, CSELT
Via G. Reiss Romoli 274,
10148 Torino, Italy
Tel: + 39 11 2169252.

1. INTRODUCTION

Two crucial elements in the development of large Expert Systems are the Domain expertise and AI expertise.

Domain Experts are usually not familiar with Expert Systems, and have only used computers as a requisite. They often find it difficult to articulate their own expertise, especially in the terms usually used by computer programmers.

That is why a knowledge engineer, experienced with computers and also a skilled communicator, is needed. He or she has to extract the expertise from the Domain Expert and from that create the Expert System knowledge.

There are two problems. Good knowledge engineers are scarce and Domain Experts are hardly available for interviews.

The ESB (Expert System Builder) will automate the process of building Expert Systems (ESs) by offering a number of powerful modelling and architectural facilities not found in previous AI-systems.

This paper first introduces the principles and tools contained in the Expert System Builder (ESB), describing the progress achieved in terms of knowledge engineering methods suitable for an industrial environment. The paper then discusses the issue of controlling non determinism in production systems and describes the Basic Expert System Builder (BESB) control architecture, which provides flexibility ease of specification for different problem solving strategies in Expert Systems.

2. AN APPROACH TO EXPERT SYSTEM BUILDING

Today Expert System Building is a complex engineering task requiring a large amount of knowledge of the internals of Expert Systems. In order to allow less experienced persons to create Expert Systems we have derived an approach to Expert System Building which is different from todays systems. Before going into more details on this approach we shall first take a look at some of the background for the selected approach.

2.1 Expert System Building

The task of building an expert system can be viewed as the mapping of the real life problem into machine representation. The real life problem often contains the following elements:

- Problem description.
  This includes both a structural and a functional description. E.g. if we want to make a diagnosis of a power plant, we must describe the power plant in terms of its structure (pumps, boilers etc.) and its functionally (creation of electricity by use of steam production).

- State description
  This includes a description of the actual state of the problem to be reasoned upon. E.g. the power plant does not produce enough electricity, and the production is continuously decreasing. This description might include many sensor based values coming from a real life application.

- Problem solving specification
  This part is usually identified as the expert knowledge. It gives declarative instructions on how problems in the problem area migth be solved. E.g. this knowledge is entered in the form of rules or predicates.

The above elements must be mapped from the users mental perception into a machine understandable form.

This mapping involves the following steps:

- Identification of the conceptual world for the problem.
  The problem must be described in a precise and consistent way to allow it to be mapped into a machine representation.

- Description of the problem structure using the defined concepts. This description can cope with the structure of the problem but normally not with the functionally aspects, which must be expressed using rules or even procedural attachments to the structural specification.

- Definition of the problem solving task:
  This requires mapping of the problem solving methods into rules and predicates. This part is required for almost all expert system builders that exist as Expert Systems usually reason by use of rules. A problem not covered by many Expert System Builders today is the specification of control in the Expert System, e.g. often the user making an expert system want to express some specific sequence in which to solve the problem.

2.2 The P96 - Expert System Builder Approach

The goal of P96 is the following:

"To investigate to which extent the building of
Expert Systems can be industrialized"

To reach this goal the general principle for the design of the Expert System Builder is to keep the mapping between the users mental model of the system and the machine representation as easy as possible. This means that we want to allow the user to be able to visualize his mental view of the system onto the machine by use of a set of graphical tools. These tools allow the user to define the model by himself, or by use of a predefined model lanquage. Further he can maintain an expert system architecture, modeling tasks and strategies, in a graphical manner allowing to model the problem solving architecture on a high abstraction level. This set of tools are supported by a system for which the conceptual universe of the domain can be defined on the machine.

Another important way to reach the goal is the reusability of knowledge already stored in the ESB. This is supported by a structuring facility for expert systems invented by this project. By use of this facility the tasks to be performed by an Expert System can be extracted and saved as a commonly available "knowledge subroutine", which can be used in other ES's needed for the same task in another context.

The complete building process for an Expert System can then be seen as a set of steps that can be iterated during maintenance of the system. Initially the ESB is empty and contains only the AI building facilities.The steps are explained in the following:

1. Definition of the domain.

   This phase is the mapping of the conceptual universe of the domain into the ESB. The domain expert has facilities to define the various objects and their attributes that exist in the domain. This includes the difinition of possible value range for attributes of the objects. As it can be seen this conceptualization requires an object oriented mapping of the users mental view, which we find is a reasonable and finite task.
   As part of the definition of the domain, the user can also define the graphical representation of the concepts so that instances of the concept later on can be visualized in a well known way.
   The primary purpose of this first step is to "teach" the ESB about the domain so it will understand the users input during the rest of the building process. Updates to this part are expected all over the Domains life time.

2. Creation of Shells.

   In this phase the domain expert defines AI reasoning elements that can be used by many expert systems - common reasoning tasks (e.g.modeling tasks, strategies, specializing object representation). Further he defines expert system shells for classes of products which later can be instantiated to form a specific product. These two parts of this phase are very important as they form the basis for rapid development of Expert Systems for a range of products.

   The creation of the reasoning elements and the ES shells makes use of graphical structure editors to build the ES architecture and knowledge editors for entering and maintenance of rules.

   Any shell or ES can be associated with a model area for modelling of the "world" on which the reasoning can take place. This model area is automatically customized to the conceptual universe during the building of the domain. By use of the model area the user can graphically create a structural model of the "world". By use of a special graphical language developed in the project it is also possible to model the functionallity of the system.

   These tools will follow the Expert System during its life time allowing to maintain structure and knowledge.

3. Creation of a Product.

   The shells defined in a domain can be instantiated to form products which can be further filled with product specific knowledge. It is normally at this stage the model of the "world" is added, as this usually is specific to the product. Often very little has to be added in order to have a running expert system based on the shells defined in the domain.

4. When a product specific expert system exists it can be applied to the problem at any time for inspection of correct functioning. As the knowledge and architecture editors forms an integrated part of any ES, the knowledge and the architecture can be modified as appropriate during the test runs.

At any phase of the building process a save of the domain and ES's to background storage can take place. This is important as the building process is an interactive task not involving any file system maintenance at the user.

At the current stage of the project this is the support the ESB can give in the building process. What is further needed is facilities to extract an Expert System to form a stand alone application, and of course to port it to a delivery environment.

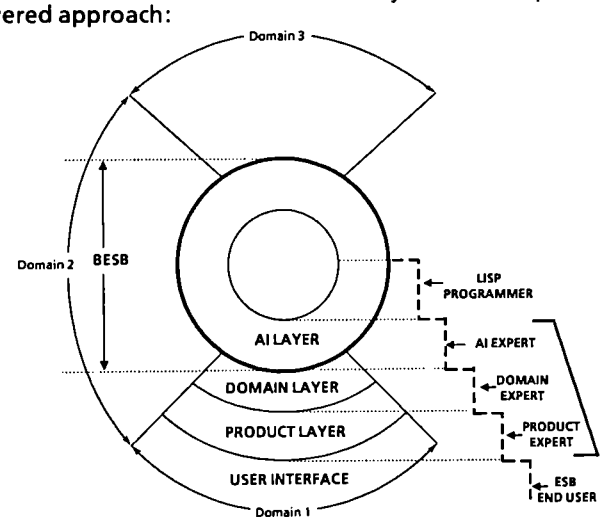
In the next section you will find an overview of the design of the ESB and some more details on the various tools.

3. OVERVIEW OF THE EXPERT SYSTEM BUILDER

This section gives a overview of the various elements of the Expert System Builder.

3.1 The layers of the ESB

In order to get a modular and well structured system the Expert System Builder is based on a layered approach:



The ESB Layers

The layers are:

- The AI layer:
  This layer provides the basic AI facilities in form of building blocks that can be mixed together to form expert systems and other "intelligent" parts of the ESB. We also name this part the Basic Expert System Builder (BESB) as it is possible to build expert system at the LISP level by using the building blocks.

- The domain layer:
  The domain layer contains the domains known by the ESB. A domain is the basis for a number of products storing the conceptual universe for the domain plus predefined ES shells.

- The product layer:
  The product layer contains the final expert systems within each domain. A product specific expert system is a system to solve a problem for a specific product, e.g. a power plant.

The BESB will be described in section 4 of this paper and the domain/product layer is covered later in this section.

3.2 The users of the ESB

In P96 we see the building of Expert Systems as a cooperation between many categories of persons each contributing to the building of the ES but on different levels of abstraction. The users are associated with the layered approach of the ESB:

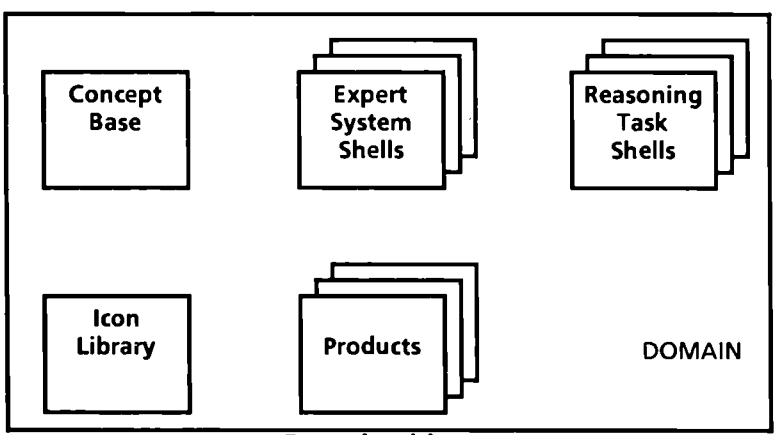
- The AI Expert:
  This person shall provide the other users of the ESB with sufficient AI tools for solving their problems. This will typically be specialisation of knowledge representation and reasoning mechanism. It is not foreseen that the AI expert shall be very active after the ESB is customized for a specific domain.

- The Domain Expert:
  This person has knowledge about the domain and solving of specific problems within the domain. The domain expert will define the conceptual universe for the domain. Further he will build Expert System elements for specific tasks which can be reused by the next level of users. He will create a number of Expert System shells already filled with knowledge for specific problems which then can be instantiated to form a final Expert System for a given product.

- The Product Expert:
  The product expert defines the actual problem for which the expert system shall be used. This means that he defines the structures and functionalities of the real world that the system shall reason about. This definition is based on the conceptual universe defined by the domain expert. Further he can add some problem solving knowledge specific for the product.

3.3 The Domain/Product layer

The design of the ESB is based on the use of object oriented programming. This influences the mapping of the layers into an implementation based on objects in the following way. We consider all "Domains" to reside in an enviroment supporting the interface towards the BESB. The figure below shows the result of this mapping:



Domain objects

The figure indicates that domain contain objects for:

- The Concept Base:
  The concept base forms the conceptual universe for the domain in form of a data base which support the creation and retrieval of domain specific concepts. The primary purpose of the concept base is to support the interfaces to the various objects in the domain with respect to terms, structure, value types etc. The concept base is common to all objects in the domain and therefore supports the reusability of domain specific knowledge, i.e. the concept base is build only once per domain and then just maintained with new concepts.

  Most concepts of the "world" can be described in terms of objects which have become the primary way of describing a concept in the ESB. Some objects will become part of a model, so to this end it is possible to define the visual representation for instances of concepts in the form of icons.

- Icon Library:
  The icon library contains the icons defined for the domain. The user can create and maintain the icon library via an icon editor, allowing to do graphic editing on the icons. A powerful set of commands support the user in that work.

- Reasoning Task Shells:
  In order to allow the user to reuse part of earlier defined expert systems the system maintains a user defined set of reasoning task shells. These shells can either be created from scratch by the user or be a kind of generalisation taken from an already existing Expert System. These shells can during the creation or maintenance of an Expert System be instantiated and inserted into the Expert System Architecture.

- Expert System Shells:
  Another level of reusability is supported by a set of user defined Expert System Shells. By shell we mean a predefined Expert System Architecture with some tasks already defined by the domain expert. The user can create an Expert System Shell by use of a set of editors viewing the Expert System architecture from different points: One is at the structural level where the user graphically can define the architecture by manipulating reasoning task into the required architecture. It shall be mentioned here that the ESB supports modularisation of the expert system in many levels both on application and on control. This is covered in more depth in section 4 and 5.

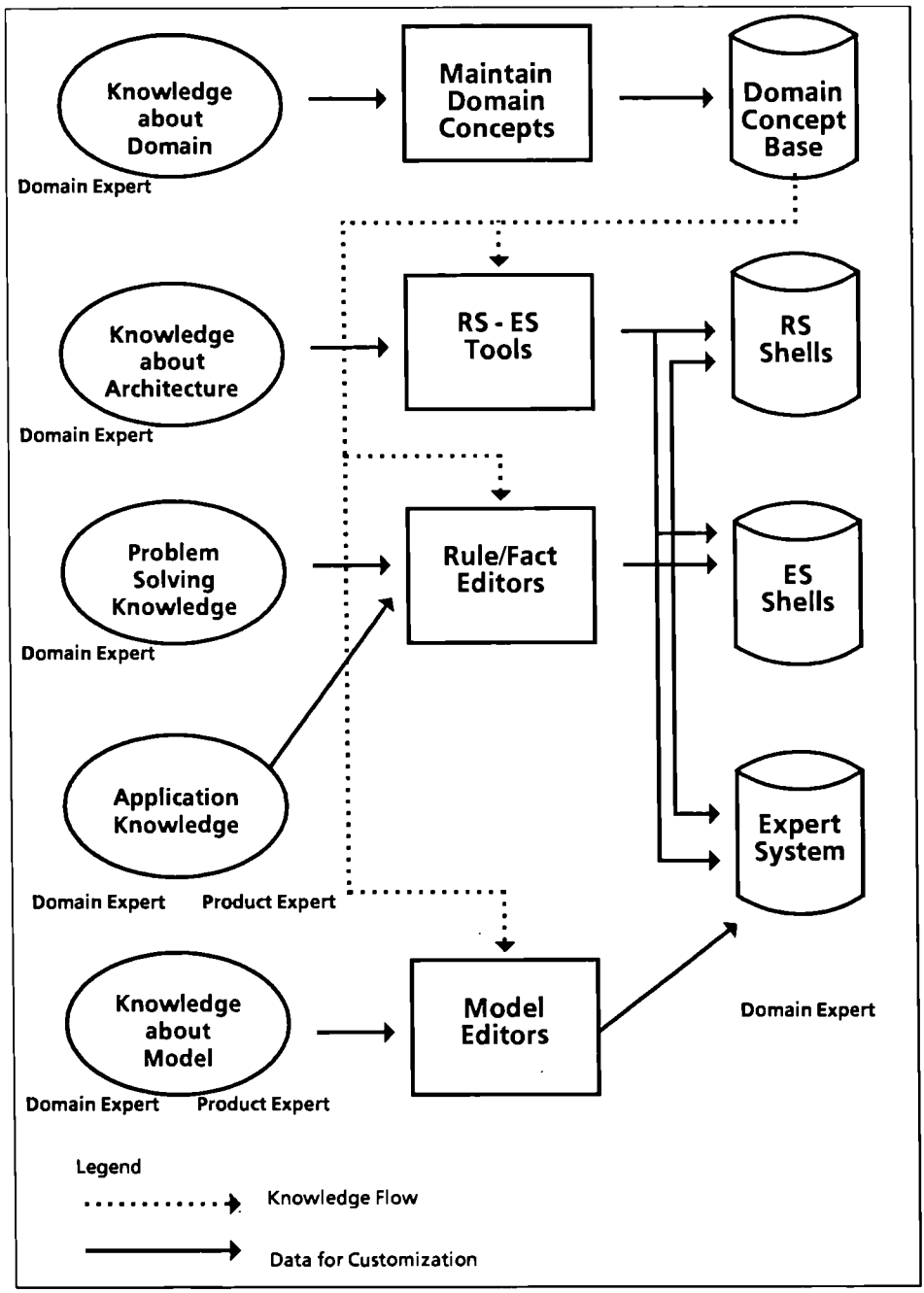
  An Expert System (and Shell) can be attached with a model area which allow for creation and maintenance of a model of the "world" which the Expert System shall reason upon. This model is created and maintained by use of a graphical editor which makes use of model elements defined as concepts in the concept base.

  The Expert System shells are intended for instantiating products which then can be specialized in further details.

- Products:
  The products are as menbtioned above instances of Expert System shells defined in the domain. The product inherits the same facilities as the Expert System Shell so the product expert can do the same manipulation with the Expert System.

All these objects are accessible through an interface guiding the user to the object which he wants to work upon. For each object a well defined set of commands are defined to manipulate the object in the proper manner.

The figure below shows the steps in the building process using these objects. It shall be noted that any step can be iterated any number of time as the application requires.



The Building Processes

4. THE BASIC EXPERT SYSTEM BUILDER (BESB)

Main goals of the BESB are a modular organization of knowledge, flexible problem solving mechanisms and a clear distinction of control knowledge from descriptive knowledge. All the above goals aim to make easier knowledge engineering by allowing clean design of knowledge bases and user control of reasoning behaviour.

Expert systems can be decomposed into modular Reasoning Systems (RSs) whose interaction achieves overall problem solving. The BESB supports the construction of complex ESs and the cooperation of different reasoning agents by means of a modular ES architecture. The critical feature is that different problem solving organizations are supported and easy to specify.

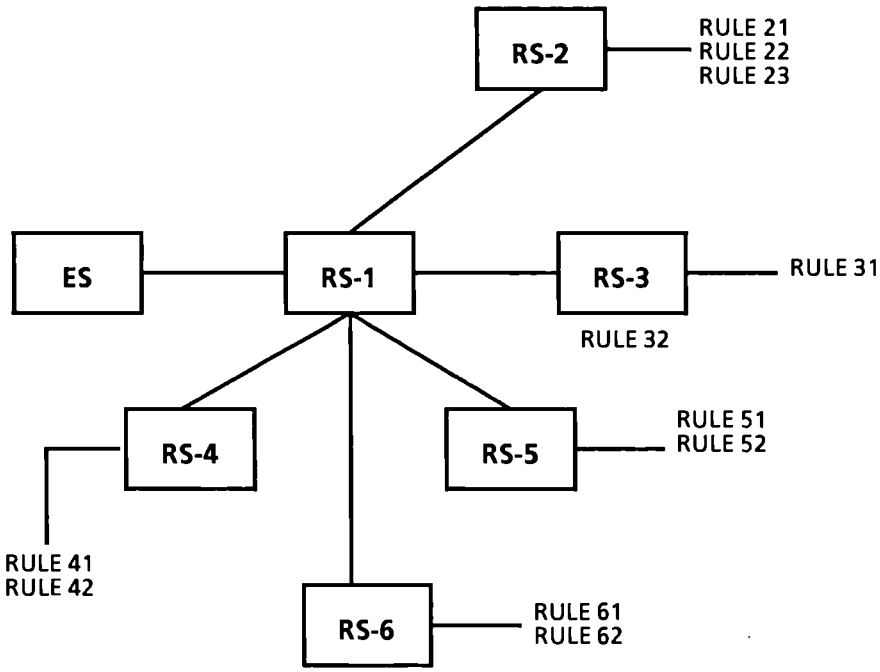
The goal of a reasoning mechanism for BESB is to use knowledge in problem solving. The Reasoning Engine (RE) of BESB provides a variety of reasoning techniques, including search and inference, backward and forward chaining. A key feature is to allow for heuristic, i.e. not only exhaustive, strategies to be specified.

Flexible strategies for inference and search are available, expressed by control knowledge, separated form descriptive (application) knowledge, to determine the behaviour of RSs. The specification of control allows a user to define different reasoning directions, conflict resolution policies and to express heuristics for making choices.

The current implementation of BESB, also referred to as BESB2, is the third prototype version of the Basic Expert System Builder. Functionalities avaible in BESB2 are described below.

4.1 Modular Architecture

ES architectures in BESB2 allow to organize any number of modules called Reasoning Systems (RSs) in a hierarchy. RSs contain rule-like operators and a structured working memory, called the Problem Space. Rule-like operators can be either a rule or an RS. This means that an RS, as any production rule, has triggering conditions and actions. At run-time an RS constitutes a node in the RSs hierachy, created by instantiated operators, as shown in the figure below:



An ES architecture

4.2 Knowledge Representation

BESB2 supports a formalism for representing knowledge as production rules, objects and facts. All entities are objects. Rules can describe patterns about both objects and facts, so that the reasoning process operates both on structured objects, and logic-like predicates e.g. (pump is stopped). The representation language can express meta knowledge, i.e. knowledge about other knowledge used in problem solving, to be used in control strategies. As we will describe in the following sections, meta knowledge describes properties of goals, states and operators, i.e. the objects of the Problem Space.

A special aspect of knowledge representation in BESB is the direct way to integrate shallow heuristic knowledge with deep knowledge describing structural and functional models of a device. This capability makes BESB a useful kernel for developing diagnostic and planning ES's.

4.3 Reasoning

BESB2 allows to reason forward from evidence to conclusions and backward from goals to preconditions. The innovative feature is that reasoning can integrate search and inference, in order to develop alternative search paths and to reason within each state. Integration is achieved when an RS performs forward reasoning by operators that either develop a tree structure of states or perform assertions within a state. When reasoning backward, operators develop a tree structure of subgoals. Both structures are explicit and users can manage through them control of the reasoning strategy, as described below. The two reasoning directions can be used in a same ES by RSs with different attributes.

4.4 Control Specification

BESB2 allows to express control strategies as attributes of an RS and to define heuristic control rules in Control RSs which can be associated to any RS. When inference and search mechanisms produce candidate GOALS, STATES and OPERATORS, their selection must be decided. In BESB2 it is possible to run either hardwired procedures for this purpose or to take decisions by means of heuristic CONTROL RULES entered by the user in a Control RSs containing rules for selecting among GOALS, STATES and OPERATORS. The Control-RS is invoked during reasoning when control problems arise, e.g. selection among candidates. The interaction of a controlled and a controlling (structurally identical) RS gives rise to an introspective capability, i.e. the capability of RSs to reason about their own behaviour and control.

The "Introspective" capability of RSs in BESB will be discussed in the next sections and compared to other research approaches on this knowledge engineering technique, important for designing complex systems.

5. CONTROL IN EXPERT SYSTEM REASONING

Expert knowledge about an application domain is the power of an expert system. An underlying reasoning mechanism, the interpreter for domain knowledge, applies the expert heuristics, often provided as production rules, while searching for the solution of the application problem. The qualitative analysis done by Stefik et al. (1) on different problem solving architectures for different uses of knowledge for problem solving suggests to adopt different problem solving architectures for different problem features. When the problem description is small, all solutions may be searched exhaustively. When the size of the problem space grows, heuristic techniques are required and search strategies can organize the space in sequences of "islands"; in complex synthesis problems planning techniques find solutions in hiearchies of abstractions. The classification considers additional requirements to handle interdependent subproblems, contradicting hypotheses, forking and joining multiple lines of reasoning, problems represented on multidimensional blackboards.

The design of a general tool, capable of helping in exploring expert system design, should support a good degree of flexibility with respect to the above problem solving architectures. A good part of the skill of knowledge engineers consists in understanding which problem solving architecture is suitable for the solution offered by an expert. This skill can be exploited by separating domain knowledge from control knowledge, guiding the behaviour of the problem solver.

The current generation of "large hybrid tolls" for building expert systems, e.g. ART [11], KEE [16] and Knowledge Craft [17], allows for rich amalgations of knowledge representation formalisms and powerful reasoning techniques. Nevertheless the issue of control specification is either severely limited or opaque and cumbersome. The design of BESB attempts to overcome limitations and complexity by providing a framework where knowledge engineers can determine a wide range of problem solving architectures by specifying control knowledge, separately from domain knowledge.

5.1 Survey

AI research investigates the issue of control knowledge, proposing the explicit representation of metaknowledge. The use of metaknowledge is surveyed by Aiello [3] covering an area of impact for this knowledge engineering technique broader than just the issue of control. A specific proposals for embedding metaknowledge in rule based systems was formulated by Davis [4] for heuristic control of productions invocation.

A key issue in effective use of metaknowledge for control is that of connecting a controlling system to a controlled system in an architecture for introspection that can inform and modify their behaviour. Introspection is the capability of a system to reason about itself. The current research on introspective systems is surveyed by Maes [5], with a discussion of:

a) which situations trigger introspection
b) what representations it manipulates and
c) how it affects the behaviour of the system.

Production systems are a widely used paradigm for representing heuristics in expert systems, therefore in the following we will focus on their interpretation mechanisms to discuss on control of reasoning. Nevertheless some of the approaches we survey are based on different paradigms.

The investigation of control issues by Laurent [2] carefully analyzes the behaviour of production interpreters and their capability to perform a wide variety of search and inference tasks under different control regimes. Typically the interpreter behaviour is affected

a) by rule invocation/selection, a control problem relevant to conflict solution in inference, and
b) by state or goal selection, a control problem relevant to search and backtracking.

Most advanced "large hybrid tools" tend to ignore the user specification of conflict resolution strategies; some provide a capability of simultaneous exploration of multiple lines of reasoning where search is not under control, as discussed by [18].

The AI approach to represent control knowledge in production systems was suggested by Davis [4] and was based on the idea of representing explicitly metaknowledge, i.e.knowledge about other knowledge prescribing how to use it. The basic idea of Davis, with the goal of controlling rules invocation, was to have metarules expressing what makes a piece of knowledge interesting in a particular context of problem solving. More precisely, metarules for controlling (base) rules invocation express what is useful or useless in the knowledge matched by (base) rules patterns and then prescribe appropriate invocation.

Other research, following the original work of Weyrauch [7], has identified the utility of using meta language: Bowen et al. in the context of logic programming [8] to avoid use of non-monotonic logic, Attardi et al. in the context of the description system Omega [9] to reason about viewpoints. A common goal of the systems above is to manage reasoning in different contexts of problem solving. The main differences are their specialized reasoning methods, their degree of flexibility and the ease of expressing domain and control knowledge.

5.2 Goals for RS Control

The above arguments led to focus BSB design on two objectives: providing RSs wtih introspective capabilities, achieving flexibility of reasoning behaviour by a control language where heuristics specified by a user prescribe how to decide on some control problems, i.e. what to do next.

In introspective systems, i.e. systems which can reason about their own state and goals, the key design issue are:

- access to representations that allow to modify behaviour, i.e. a communication problem.

- a mechanism that allows domain reasoning to interact with control reasoning, i.e. a control problem.

The actual capabilities of introspection determine the flexibility of a problem solver and ease of expressing control knowledge.

In BESB we aim at modelling the architecture on limited control issues for an RS. The typical control issues encountered in expert system deal with selection of goals, states and operators within a reasoning agent. BESB provides users with a control language for deciding when selecting goals, states and operators. The expected benefit is to be found in the clarity of representing control knowledge in ES prototypes, either for rapid experimentation of diverse problem solving behaviours, or for building large rule bases.

5.2.1 Control Flow in the Reasoning Engine

The BESB engine for using knowledge in problem solving is called Reasoning Engine (RE). As a starting point we have integrated in the RE the functionality of search and inference within a uniform mechanism. An interesting inference analysis of problem solving paradigms was developed by Simon [10] in order to exploit the commonalities and differences of search and inference metaphors (Simon uses the term reasoning where we use inference). Simon characterizes inference as a process of knowledge "accumulation" in search of a proof, while characterizing search as evaluation of alternatives inferred by a concise representation of problem states.

The two metaphors are somehow complementary and non exclusive; BESB adopts both as primitive problem solving mechanisms in the RE. The RE uses productions to perform a) logic-like inferences and b) to create a search space of alternatives. When we use inference we intend to have means for adding new knowledge, e.g. asserting new facts or creating subgoals. When we use search we intend to have means for exploring alternatives, i.e. finding a possible path to a solution. Integrating the two means to exploit when knowledge is useful during exploration sharing a common framework.

Integration of search and inference is not original per se.

The original feature is that the structures internally developed by the RE are accessible for inspection and modification by metareasoning. The RE represents an object, the Problem Space, storing candidates for decisions, i.e. provable goals, expandible states and applicable operators. A Control RS is then reasoning about such "controlled" objects.

The Reasoning Engine goes through a match/select/execute cycle, where the non deterministic "select" computation is to choose one out of possible many candidates, a typical situation in search and, in inference, originated by pattern matching and unification. Selection of goals, states and operators, embedded in the match/select/execute cycle, provides a model of the reasoning process, while the Problem Space provides a representation of the reasoning enviroment. These two aspects are generelly crucial for introspective systems [5], since metareasoning about a computational processs requires an accessible representation of the enviroment and continuation of the process. Once a decision is taken the cycle continues applying search/inference operators.

Decisions about selections can be taken via hardwired methods, e.g. a straightforward depth first search procedure or via metareasoning. In the latter case another reasoning process, described in a later section, uses heuristic rules written by the user to decide. Note that the effect of hardwired or metalevel control are the same, i.e. to update the Problem Space and resume the basic cycle. This uniformity of representation allows the interleaving of the two control regimes.

Execution of the selected operator causes assertion and retraction of new facts, whose effect will incrementally update a compiled representation of rule patterns and their matches.

5.2.2 Reasoning Strategies

A few user-defined attributes specify the behaviour of RSs: direction, reasoning mode, search strategy and conflict resolution strategy. From the combination of their values arise diverse problem solving strategies. Direction can be forward or backward, reasoning mode can be inference, search or reasoning, i.e. the integration of the two. Search and conflict-resolution can be either hardwired procedures or "RS", i.e. based on control reasoning.

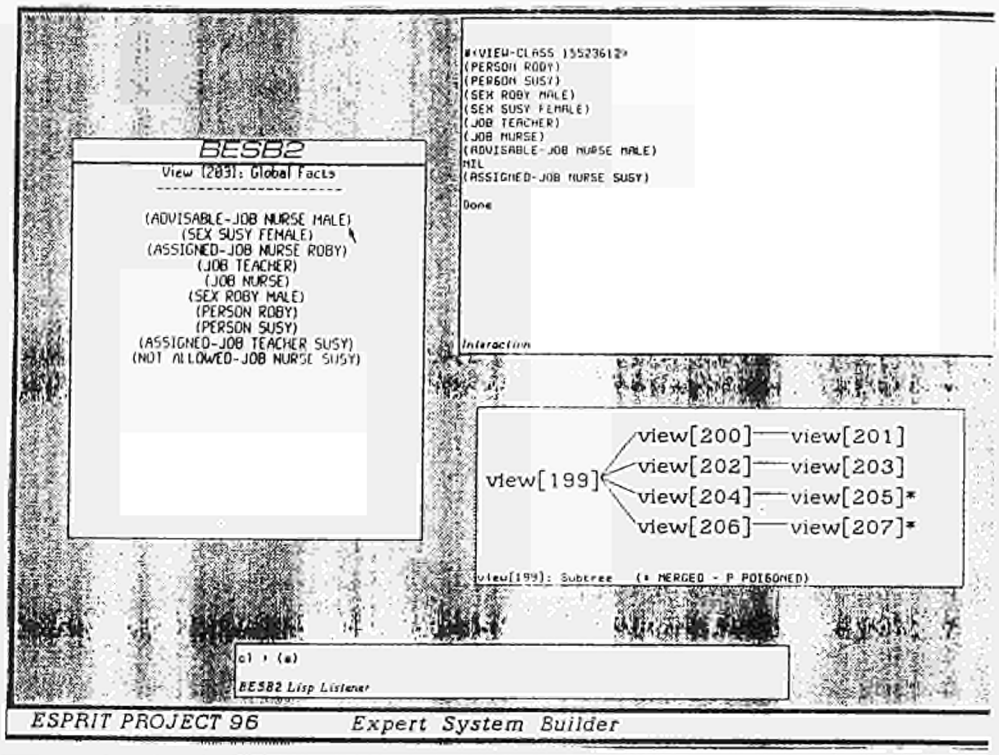
In this section we give a simple example of how the third reasoning mode can significantly prune search by an early posting of constraints. For other problems it would be suitable to use an opposite strategy, i.e. delaying the posting of constraints. This is not difficult in BESB, since it is enough to specify, via the conflict-resolution strategy, whether we want to schedule search operators before inference operators (here used for representing constraints), or the other way around.

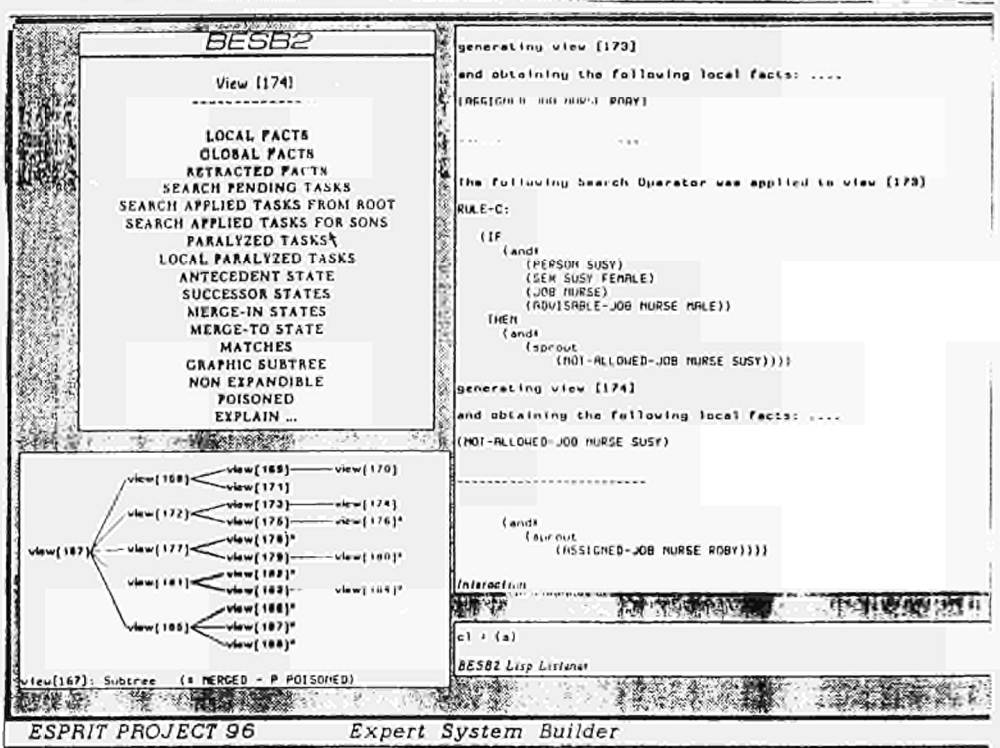
The example aims to solve the following problem suggested by Wos [19]: N persons must be allocated one out of N jobs, under the constraint that some of the jobs are advisable for persons of a given sex. Find a job allocation respecting the constraint. The problem gives rise to a large number of alternatives; furthermore, once a job is allocated, the state of the world changes, therefore if we represent the knowledge about a job allocation to a person by a rule and the constraint by another rule, a monotonic inference process may fail to reach a solution. So the normal way to reason about this problem is to search for all possible rule sequences, which quarantees that a solution will be found. Nevertheless a blind search process may explore unnecessary paths to the solution. In conclusion the simple strategy that we suggest is to represent:

- by a search operator knowledge about job allocation, in order to span all possible job-person pairs;

- by an inference operator knowledge about the constraint which prevents to allocate a job to a person of the unadvisable sex;

The reasoning strategy will be specified in order to apply the inference operator before the search operator, resulting in a reduced search space, since only the legal alternatives would be expanded. The two figures below show the search space generated by the blind search strategy and the one generated by the "reasoning" strategy for the simply case of two persons.


BESB2
View [203]: Global Facts
(ADVISABLE-JOB NURSE MALE)
(SEX SUSY FEMALE)
(ASSIGNED-JOB NURSE ROBY)
(JOB TEACHER)
(JOB NURSE)
(SEX ROBY MALE)
(PERSON ROBY)
(PERSON SUSY)
(ASSIGNED-JOB TEACHER SUSY)
(NOT ALLOWED-JOB NURSE SUSY)
#<VIEW-CLASS 15523612>
(PERSON ROBY)
(PERSON SUSY)
(SEX ROBY MALE)
(SEX SUSY FEMALE)
(JOB TEACHER)
(JOB NURSE)
(ADVISABLE-JOB NURSE MALE)
NIL
(ASSIGNED-JOB NURSE SUSY)
Done
Interaction
view[199]
view[200]
view[201]
view[202]
view[203]
view[204]
view[205]*
view[206]
view[207]*
view[199]: Subtree (* MERGED - P POISONED)
cl > (a)
BESB2 Lisp Listener
ESPRIT PROJECT 96
Expert System Builder


BESB2
View [174]
LOCAL FACTS
GLOBAL FACTS
RETRACTED FACTS
SEARCH PENDING TASKS
SEARCH APPLIED TASKS FROM ROOT
SEARCH APPLIED TASKS FOR SONS
PARALYZED TASKS
LOCAL PARALYZED TASKS
ANTECEDENT STATE
SUCCESSOR STATES
MERGE-IN STATES
MERGE-TO STATE
MATCHES
GRAPHIC SUBTREE
NON EXPANDIBLE
POISONED
EXPLAIN ...
generating view [173]
and obtaining the following local facts: ....
[ASSIGNED-JOB NURSE ROBY]
[the following Search Operator was applied to view [173]
RULE-C:
(IF
(and*
(PERSON SUSY)
(SEX SUSY FEMALE)
(JOB NURSE)
(ADVISABLE-JOB NURSE MALE))
THEN
(and*
(sprout
(NOT-ALLOWED-JOB NURSE SUSY))))
generating view [174]
and obtaining the following local facts: ....
(NOT-ALLOWED-JOB NURSE SUSY)
(and*
(sprout
(ASSIGNED-JOB NURSE ROBY))))
Interaction
view[167]: Subtree (* MERGED - P POISONED)
cl > (a)
BESB2 Lisp Listener
ESPRIT PROJECT 96
Expert System Builder

5.3 Control Reasoning

In order to describe how control reasoning is designed in BESB, we will first overview its architecture, then analyze how communication is ensured between a controlling and a controlled RS, define the means for invoking control reasoning and conclude by outlining work now in progress.

5.3.1 Control Architecture

Two main problems are considered in the RE in order to provide an introspective architecture:

- a control problem, i.e. allowing decisions about candidate goals, states and operators selection, triggering a jump to the meta level;

- a communication problem, i.e. connecting the Problem Space of the controlled RS to the reasoning process of the Control RS.

We introduce the control problem by discussing architectural concepts taken from the SOAR system [12]. The introspective architecture of SOAR recursively calls the interpreter for each problem that cannot be solved at one level of interpretation creating a new subgoal. Eventually SOAR distinquishes among automatic subgoaling, i.e. when the interpreter does not know how to solve an impasse and uses weak methods at the recursive level, and deliberate subgoaling, i.e. when a subgoal is created directly in order to use heuristic capable of solving it.

In BESB we are interested in using heuristics entered for the deliberate user purpose of controlling decision; metareasoning is only possible when a Control RS is created and related to the controlled RS. If the user specifies hardwired control strategies, e.g. depth first search as in the example of the previous section, a procedure pops a state from the stack of expandible states. If the user specifies heuristic control strategies, the controlled RS invokes its Control RS. At each decision point in the RE match/select/execute cycle, it is possible to jump to the meta level and resume when the Control RS has made a choice. This leads us to consider the communication problem.

The rationale for the communication problem originates from the fact that typically control heuristic specify that some of the candidates to selection is valuable by describing some interesting property of its contents. This idea was originally proposed by Davis [4] and described as "content reference". Fig. 1 shows an example of a BESB control rule identifying the relevance of a state containing a desired pattern of knowledge (a "yes-predicate"). The rule prescribes to select the state with a related matching operator.

```
(defrule select-state-and-operator
    (and*
        (expandible ?state)
        (applicable ?state ?operator ?bindings)
        (state ?state (yes-predicate ?var-x ?var-y))
        (expandible ?state1)
        (applicable ?state1 ?operator1 ?bindings1)
        (state ?state1 (no-predicate ?var1-x ?var1-y)))
    (selected ?state ?operator ?bindings))
```

- Heuristic rule about contents of states -

The key points for an introspective control architecture can be stated as follows:

- the RE at each cycle updates the Problem Space and inputs a new situation to the Control RS, whose goal is to decide "what to do next";

- the control rules match against the updated situation accessing the contents of the controlled Problem Space.

The efficiency of the resulting behaviour depends for the latter aspect by the capability of reasoning about obejcts and for the former by an incremental compilation of changes occurred in the controlled RS onto the patterns of the control rules.

When the Reasoning Engine activates the Control RS, it communicates the set of candidate goals, states or operators currently available for the choice; when resuming, the Control RS answers with the selected element. The controlling and the controlled system thus communicate by means of messages implemented as "control facts", e.g. facts telling which instance of operator is applicable in an expandible state. The two interacting systems exchange declarative facts between them, i.e. statements such as:

(applicable ?state ?operator ?binding)

or

(selected ?state ?operator ?binding)

which connect the controlled Problem Space to the controlled RE. They are dynamically created when the two systems need communication and are the key to allow control rules to refer to the contents of the objects to be selected.

As a concluding remark, let us note that there can be any number of levels of meta reasoning, but the user can determine a simple criterion for deciding at the highest level he has specified, possibly the first.

5.3.2 Meta level Sensors and Effectors

The communication problem is to provide practical means to inform the controlling system and modify the controlled system, i.e. to allow metareasoning to be informed by and to affect domain reasoning. The issue of "causally connecting" the levels in an introspective system was introduced by Batali [13] in his investigation, and can be simply stated as the need to provide access capabilities affecting the resulting behaviour. Access from the meta level is directed to "sense" the controlled environment, i.e. the Problem Space and its contents. Design options for connecting the metalevel are discussed by Genesereth in [14]. For "sensing" the base level from the meta level one can use "semantic attachments" as in FOL [7]. Furthermore the base level may be extended to update the meta level.

For this purpose BESB provides specific meta level sensors reaching object in the base level. The previous rule selecting state and operator shows the pattern

1) (state ?state (yes-predicate ?var-x ?var-y))

which looks for the pattern

2) (yes-predicate ?var-x ?var-y)

inside any of the states which match the pattern

3) (expandible ?state).

The"state" pattern in 1) is a meta level "sensor", informing the Control RS about the contents of states belonging to the controlled RS.

5.3.3. Control Invocation

BESS allows an adaptable method for control invocation, which enhances the one described so far. Attributes specifying how a RS behaves can select either a "hardwired" procedural strategy or a Control RS for heuristic strategies. Since the use of heuristic strategies may only be relevant to some specific situation, the new BESB design allows to interleave a default procedural strategy with the one provided by a Control RS, triggered by the presence of a desired situation. The adaptable mechanism is justified by two different reasons:

- to limit the overhead of Control RS computations;
- to allow a user to specify critical situations, and associated heuristics for making decisions, occuring in the "routine work" of problem solving.

The capability of adaptable control seems to capture the essence of a strategy, since a strategy aims at forecasting critical decisions points and at making relevant knowledge to be available exactly when needed.

6. CONCLUDING REMARKS

The P96 project is currently in its fifth year of the five year contract. At this stage the third version of the BESB is running together with the first version of the ESB. The ESB has been used to build a number of sample expert systems for testing purposes.

Currently an "industrialization" phase is taking place in order to build a more robust and stable system, which can form the basis for better competition against Japan and USA.

A "big" demonstrator Expert System is beeing build in parallel with this "industrialization" in order to make a complete testing af the facilities of the ESB. This demonstrator is a complete online diagnosis system for the Boiler system of a power plant.

7. REFERENCES

1. Stefik, M. et al. "The Organization of Expert Systems, a Tutorial" Artificial Intelligence 18 (1982)
2. Laurent, J. P. "Control Structures in Expert Systems" Technology and Science of Information, vol. 3, nr. 3 (1984)
3. Aiello, L. "The Uses of Metaknowledge in AI Systems" Proc. ECAI 84: Advances in AI - Elsevier Science (1984)
4. Davis, R. "Meta-rules: Reasoning about Control" Artificial Intelligence 15 (1980)
5. Maes, P. "Introspection in Knowledge Representation" Proc. ECAI 86.
6. Jensen, F. R. "Flame Reference Chart" ESPRIT Project P96, Report 381-WP-03204-535.
7. Weyrauch, R. W. "Prolegomena to a theory of Mechanized Formal Reasoning" Artificial Intelligence 13 (1980)

8. Bowen, K. A. "Amalgamating Language and Metalanguage" Logic Programming, Academic Press (1982)

9. Attardi, G et al. "Metalanguage and Reasoning across Viewpoints" Proc. ECAI

10. Simon, H. A. "Search and Reasoning in Problem Solving" Artificial Intelligence 21 (1983)

11. "Inference ART Reference Manual" Inference Corporation, Los Angeles, California (1986)

12. Laird, J. E. "Universal Subgoaling" Carnegie Mellon University, Department of Computer Science, Technical Report CMU-CS-84-129, Pittsburgh, Pennsilvanya (1984)

13. Batali, J. "Computational Introspection" Massachusetts Institute of Technology, Artificial Intelligence Laboratory, AI Memo no. 701, Cambridge Massachusetts (1983)

14. Genesereth, M. R. "An Overview of Metalevel Architecture" Proc. AAAI 83

15. McDermott, D. "Generalizing Problem Reduction: a Logical Analysis" Proc. 8.th IJCAI Karlsruhe (1983)

16. "KEE, the Knowledge Engineering Environment" Intellicorp, Menlo Park CA

17. "Knowledge Craft, an Environment for Developing Knowledge Based System" Carnegie Group, Pittsbourg PA

18. De Kleer, J. et al. "Back to Backtracking: Controlling the ATMS" Proc. AAAI 86

19. Wos, l. et al. "Automated Reasoning: Introduction and Applications" Chapter 3, section 3.1. Prentice Hall, Englewood Cliffs NJ.

Project No. 387

INDUSTRIAL CONTROL : A CHALLENGE FOR THE APPLICATIONS OF ARTIFICIAL INTELLIGENCE

Some Lessons Learnt and Results from the KRITIC Esprit Project (P387)

Authors : F. Arlabosse (+), J. Biermann (++), E. Gaussens (+), T. Wittig (++)

(+) FRAMENTEC S.A. (France)
(++) KRUPP ATLAS ELEKTRONIK GMBH (West Germany)

ABSTRACT

Based on the work conducted in the framework of the KRITIC Esprit Project (P387), some results concerning the application of knowledge-based systems to industrial control are described.

Two main aspects are dealt with :

1. The importance of knowledge-based system control, to address this kind of application.

   Two significant illustrations of this are :

   *how control engineers are effectively using a representation of their actions and abstract control for their decision processes ;

   *how the latter must be reproduced by any KBS application in industrial control.

   This methodological attempt is one of the achievement of the KRITIC project.

   Based upon detailed analysis, two viewpoints have been developed inside the project team :

   *classification

   *contraints satisfaction search

   These are sketched in the last part of CHAPTER ONE.

2. The construction of a KBS for load management in electrical power stations based on constraint satisfaction search.

   This is illustrated by a planning system used in the demonstrator built by Krupp Atlas Elektronik (KAE).

   This part of the paper also briefly describes various tools built in the course of the KRITIC project.

   The classisfication viewpoint and its application to the diagnosis of a telecommunications switching system is described in [1].

To conclude, the results of the KRITIC project are summarized, in particular :

. A methodological framework for KBS development in industrial control applications ;

. The architectural and control specifications coming from that methodology ;

. And finally , that the decision processes involved solve problems by using heuristics which, in some cases, either are synthetized or "shortcut" mathematical algorithms of automatic and optimal control theories.

ACKNOWLEDGMENTS

The autors would like to acknowledge many valuable discussions and sharing of of experience with :

The Queen Mary College KRITIC Team :
E. Mamdani, J. Bigham, V. Khong, S. Varey ;

The British Telecom KRITIC Team :
G. Williamson, J. Butler, S. King ;

And, from Framentec :
V. Duong, P. Le Page.

INTRODUCTION

Based on the work conducted in the framework of the KRITIC Esprit Project (P387), some results concerning the application of knowledge-based systems to industrial control are described.

Two main aspects are dealt with :

1. The importance of knowledge-based system control, to address this kind of application.

   Two significant illustrations of this are :

   *how control engineers are effectively using a representation of their actions and abstract control for their decision processes ;

   *how the latter must be reproduced by any KBS application in industrial control.

   This methodological attempt is one of the achievement of the KRITIC project.

   Based upon detailed analysis, two viewpoints have been developed inside the project team :

   *classification
   *constraints satisfaction search

   These are sketched in the last part of CHAPTER ONE.

2. The construction of a KBS for load management in electrical power stations based on constraint satisfaction search.

   This is illustrated by a planning system used in the demonstrator built by

Krupp Atlas Elektronik (KAE).

This part of the paper also briefly describes various tools built in the course of the KRITIC project.

The classification viewpoint and its application to the diagnosis of a telecommunications switching system is described in [1].

To conclude, the results of the KRITIC project are summarized, in particular :

. A methodological framework for KBS development in industrial control applications ;

. The architectural and control specifications coming from that methodology ;

. And finally, that the decision processes involved solve problems by using heuristics which, in some cases, either are synthetized or "shortcut" mathematical algorithms of automatic and optimal control theories.

ACKNOWLEDGMENTS

The authors would like to acknowledge many valuable discussions and sharing of experience with :

The Queen Mary College KRITIC Team :
E. Mamdani, J. Bigham, V. Khong, S. Varey ;

The British Telecom KRITIC Team :
G. Williamson, J. Butler, S. King ;

And, from Framentec :
V. Duong, P. Le Page.

CHAPTER ONE : CONTROL OF KNOWLEDGE-BASED SYSTEM FOR INDUSTRIAL CONTROL AN ESSENTIAL ISSUE

One of the achievement of the KRITIC project was to highlight the role that can be played by knowledge-based systems in industrial control applications.

The relatively recent penetration of knowledge engineering techniques in the field of industrial control raises new problems for the artificial intelligence (AI) researchers and developers.

This is because of the complexity of the problems to be solved and the many "classical" approaches to solutions, from pure mathematic fields (control theory, differential games, dynamics) to purely technical areas (sensors, actuators, automatic control devices), all of them often being integrated into a computer structure and embedded in the very general problem of "decision-making" in non-nominal or even critical situations.

The use of knowledge engineering techniques in this field is more or less justified by the fact that control "decisions" are based in fact on human experience and judgement, using the variety of techniques, designs or theories previously mentioned.

Also, the knowledge used by industrial process control engineers keeps evolving as their experience with the industrial system grows.

In some ways, the KRITIC project is an attempt to study this decision-making process and build AI tools to demonstrate how parts of it can benefit from this new approach.

The concrete examples analysed are a telecommunication switching system and load management of an electrical power distribution network.

1] Some Basic Characteristics of Industrial Process Control

Industrial control means the way a process is supervised and kept within its operating limits.

Thus, industrial process control (IPC) systems need to be able to evaluate a situation, then make the appropriate decisions.

The situation is theoretically defined by the process state, given by sensors, and formal models of the design, complemented by exact knowledge of the relevant environment.

In practice, however, it is evaluated through a partially known/controllable environment, and a REPRESENTATION of the process based on : previous actions/reactions, encompassing but synthetic information (e.g., a unique integrated curve of consomption for a whole network), and finally, the engineer's experience (induction/deduction - taxonomy - informal models - personal notes, etc...).

In general terms, a control system must be able to recognise dangerous or potentially dangerous situations, to be capable of optimizing a situation (or its evolution), to plan a sequence of actions to avoid a critical situation, to plan a sequence of action to escape a critical situation, and finally, to plan a sequence of actions to optimize (the evolution of) a situation.

The process representation provides the basic background knowledge used by the process control engineer.

Since process representations are, as we have seen, dependent on the actions, the synthetic information and the heuristics, it is essential that they are kept coherent as the process goes on. This implies that the engineer cannot take isolated decisions ; they have to be incorporated into a plan, to avoid destroying the informal model being used.

Somehow this model acts as a cohercive solving process for the decision model.

Industrial process control system operation is based on :

a - building a sequence of actions relying on the representation(s) and their evolution(s) ;

b - being able to reason and select a solution to a problem that is under-constrained, either because of unpredicted behavior of the process or of the environment, or due to the imprecision of the plausible representation(s) ;

c - being able to relax a problem that is over-constrained either because of a crisis situation or due to the rigidity of the representation(s) ;

d - performing these actions and comparing the results with the available observations (sensors, or synthetic process curve, etc...) ;

e - in case of discrepancies or contradictions that lead to risky situations, taking the appropriate remedial actions immediatly ; and

f - simultaneously replenning future actions, to keep the consistency of the overall sequences and of the various representations.

This implies that the various problems need to be solved under a coherent and encompassing framework or problem solving scheme : at the action level, at the action-planning level and at the model/representation/plan level.

Although classical approaches are often very efficient, it is plausible to believe that in case of "open processes", dealing with such external inputs as the actions of power consumers, the reasoning heuristics, as well as the repretational modes of the control, give to AI techniques an opportunity to demonstrate their usefulness (not in a "stand-alone" fashion but integrated with other appropriate techniques).

2] Some Basic Features of KBS Control

To introduce our analysis, we might think of control as being the means by which problem solving strategies are selected, and even further, by which problem-solving methods to use in the face of a situation are choosen. Let us quote B. Hayes-Roth [2] :

"The control problem is : which of its potential actions should an AI system perform at each point in the problem solving process ? [...]
in solving the control problem, a system determines its own cognitive behavior. [...]
people do not simply solve a problem. They often know something about how they solve the problem, how they have solved similar problems in the past, why they perform one problem solving action rather than another, what problem-solving actions they are likely to perform in the future, and so forth."

This clearly points out what process control using KBS has to achieve, namely use the problem-solving knowledge to operate the knowledge-based system itself, which in a way creates a new problem to solve.

Obviously this is not easy, since, as stated by de Kleer-Doyle-Steele-Sussman [3] :

"Verily, as much knowledge is needed to effectively use a fact as there is in the fact".

If we think about control in terms of the classical "fetch-execute" cycle in computers and its interpreter-agenda formalization, then all the knowledge mentioned above should be incorporated into the interpreter.

Then we face the problem quoted by Hayes [4] :

"We have now come full circle, to a classical problem-solving situation. How can the interpreter decide what order to run the process in ? it doesn't know anything about any particular domain, so it can't decide. So we have to be able to tell it. [...] This is exactly the situation which [...] proceduralists attacked. In removing the decision to actually RUN from the code and placing it in the interpreter, advocates of [agenda structures] [...] have recreated the uniform black-box problem-solver."

To overcome this problem, several solutions are proposed : define a standard order for running tasks (priorities), define a semiadaptative scheme for runring tasks (set priorities and "find proper knowledge sources based on facts, sort them according to fixed priorities, run the first one, but no heuristics

are incorporated into each basic action of the cycle"), or even define, like Lenat [6] totally dynamic priorities based on the accumulation of reasons to run a task.

However, still citing Hayes analysis [4] :

"A somewhat more sophisticated idea is to allow descriptors for subqueues and allow processes to access these descriptors.
But none of these ideas seem very convincing. And we have moved down another level, to the interpreter of the interpreter-writing language of the representation language.

The only way out of this descending spiral is upwards.
We need to be able to describe processing strategies in a language at least as rich as that in which we describe the external domains, and for good engineering, it should be the same language."

So, since control is a problem overlaying other problems we once again face the difficulties in qualifying its attributes and goals, enhanced by the inherent complexity of the notions of "meta levels" (see eg [5], [6], [7].

like any problem-solving framework, the control problem has three facets :

- objectives of the control : the problem to be solved

- behavioral goals of the control

- representation of the control solving strategies.

The last facet is tackled within part 3 and chapter 2.

A very clear description of the behavioral goals is given by B. Hayes-Roth in [2].

In our systems, the control objectives are split into three levels : the inference strategies level (level 1), the resolution steps coordination for a specific problem (level 2), and finally the choices on what problems to solve now (level 3).

To illustrate this taxonomy, let us go back to our industrial process control applications.

As we have seen in part 1, the various overall problems to be solved can be grouped into three classes : plan actions, act on the system, observe its reactions.

In a KBS for such applications, a "problem-solving module" will be attached to each of these problems.

The first decisions to be taken are then : what module to run now, if such is the situation, what modules to run at the same time (multitasking, multiprocessing) ? Together with the management of requests to external resources (databases, knowledge bases, etc...), these decisions are the ones to be taken in level 3.

Having decided to run, for example, the planning module, control level 2 is targeted toward coordinating the different steps (or "planning islands" in the Minsky termonology [21], to achieve the goal assigned at the higher level.

Level 2 might have to solve problems such as : loop detection, coherency management, consistency checking, backtracking, completeness, contraints relaxation, etc... Inside each step, one has to decide what ruleset to use, with which strategy, eventually to select backward or forward chaining modes, in order to achieve the goals related to it.

These problems are solved at the lowest level of control, namely level 1.

The way this taxonomy could be implemented is explained in chapter 2.

3] Classification and/or Constraint Satisfaction Search Viewpoints ?

The process control architecture could be defined by using Agenda mechanisms in each of the layers, following the B. Hayes-Roth proposals.

This might sound very useful in case where control mechanisms should behave opportunistically, and when the industrial process control mechanisms are innovative most of the time, or when the flow of information coming from the process very often contradicts the actual reasoning.

In these the cases, KBS control should reproduce this flexibility by giving a high degree of freedom in the sequences of choices of actions.

Basically, it is then a classification problem using multiple modes of classifications (depending on the abstraction needed), based on a single percolation mechanism, possibly specialized.

This viewpoint comes from diagnostic problems, or planning of experiments (Stefik [8], speech understanding (HASP [9], or finally multitask planning, where the problem is to reconcile constrained schedules with unconstrained actions, such as errands (OPM, B. Hayes-Roth [2]).

The overall architecture could be defined by explicitely separating the layers as objects, and explicitzly defining the classification-percolation mechanisms inside each object.

Another, more traditional approach is to use a single classification (i.e., a single blackboard) linked to the domain and providing the input or justification for the control decisions. This approach has the potential danger noted by Hayes (see [2]).

In the KRITIC project these approaches have been followed in the Telecommunication switching domain (which is essentially a diagnostic problem).

Another approach, used for the other application studied in the KRITIC project, based on constraint satisfaction, is developed in CHAPTER TWO.

This approach could be integrated, at the conceptual level with the works of Lansky [10] and Georgeff [11].

The practical approach being followed is to formalize the analysis of the application, emphasising the dynamic, time-dependent links within each module, providing a descript of the required synchronization.

CHAPTER TWO : LOAD MANAGEMENT PLANNING USING A KNOWLEDGE-BASED "EXPERT SYSTEM"

1] Introduction

This chapter describes the planning component of an expert system for assisting in the load management of an electrical power distribution network. Load management is important for electrical power generation and distribution to maintain an optimum load for the overall system. Instead of performing ad hoc-actions whenever the need arises (cutting off certain consumers or increasing power generation), a plan for tacking such actions is usually prepared in advance.

This planning can be done using conventional techniques, but since it is difficult to incorporate uncertain knowledge - such as consumer behavior - into such algorithms, an expert system approach is suggested.

The expert system described here is a prototype, not yet connected to a real power distribution utility, but operating with a simulation of the load behavior of a full-size distribution net. In this test-bed, the system operates in closed-loop fashion, without any simulated operator interventions. We want to state very clearly that such closed-loop operation is not foreseen when this system is actually installed in a plant control room. Too many issues of knowledge-base integrity and security have to be solved first, before any closed loop operation can possibly be considered.

After having described the implemented application, we shall stress in part 6 the correspondance with the general descriptions of CHAPTER ONE.

2] Load Management

Among the many tasks that the control-room personnel have to accomplish, such as maintenance of line fault tracing, load management is one of the most important, since it has a direct impact on the economic results of the electrical utility. From the point of view of the work described here, there are two fairly different structures of such utilities. One is really pure distribution, in the sense that these utilities buy all of their electricity from outside suppliers. The only thing they have to be concerned with is not to exceed a given limit of energy, as described in more detail later on. The other type of distribution utility - besides buying some electricity from outside sources, also run their own power stations to back up peak demand or to supply the base load. So they have the further load management objective of economically operating their power stations.

Figure 2-1 shows a typical load curve of a large distribution network on a cold winter day with its peak areas at breakfast, lunch, and dinner time. The sharp increases and decreases around 6:00 h and 22:00 h are due to the switching on and off of two major storage-heater groups, respectively.

2.1.] Main Constraints

Typically, an electrical utility distributing power to its customer has to operate under the following basic constraints :

There is a strict maximum limit, which under no circumstances must be exceeded. This limit is determined by the contract between the distribution utility and the electricity supplier. Its level determines the cost of the electricity to the distributor.
The complete day (24 hrs) is divided into 15-minute time slots (96 in all).

At the end of every time slot the total energy consumed is calculated and compared against the predetermined maximum limit. Exceeding this limit invokes high supplementary costs for the distribution company (contractual penalty), which are not recoverable from their consumers.

. To reduce the overall cost it is desirable to minimize the margin between the allowable limit and the actual consumption.
Since the overall consumption, i.e. the integral over a day, cannot be influenced by the distribution utilities, the only way to achieve this aim is to try to shift certain consumers into low demand times and thereby flatten the load-curve over the full day.

If the operator considers, or judges from experience, that the maximum limit will be exceeded, there are certain actions that he can take to avert this. Because the energy used is measured every 15 minutes, it is possible to take evasive action. If, on the other hand, the instantaneous power were measured there would be no question of controlling the process, as the system response can be considered to be quite slow. The process of taking evasive action is highly dependent on the human operator and hence is an area where knowledge-based techniques could prove very useful.

3] Objectives and Means

Based on the previously mentioned constraints, three main objective of load management can be identified :

a - Prevent the consumed power from exceeding a certain (dynamic) limit ;

b - Optimize the actual power consumption, i.e. shift power consumption from peak load times to low demand times ; and

c - Satisfy the various tariff conditions (e.g. some consumers are only allowed to be switched on during nighttime, i.e. in the low-tariff period).

To achieve these objectives, the control room staff has certain ways to influce the power consumption. These are essentially switching some groups of special sonsumers on or off during some times of the day. Typical consumers that may be centrally controlled are :

- industrial consumers

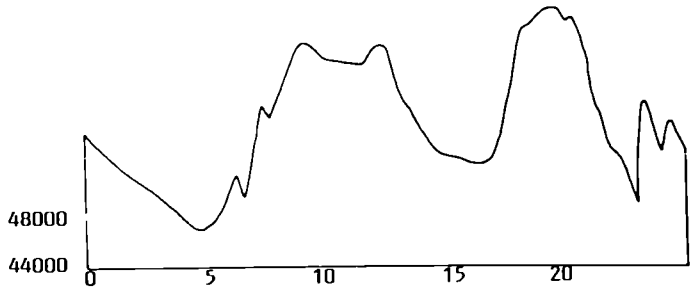
- storage heaters

- direct heaters

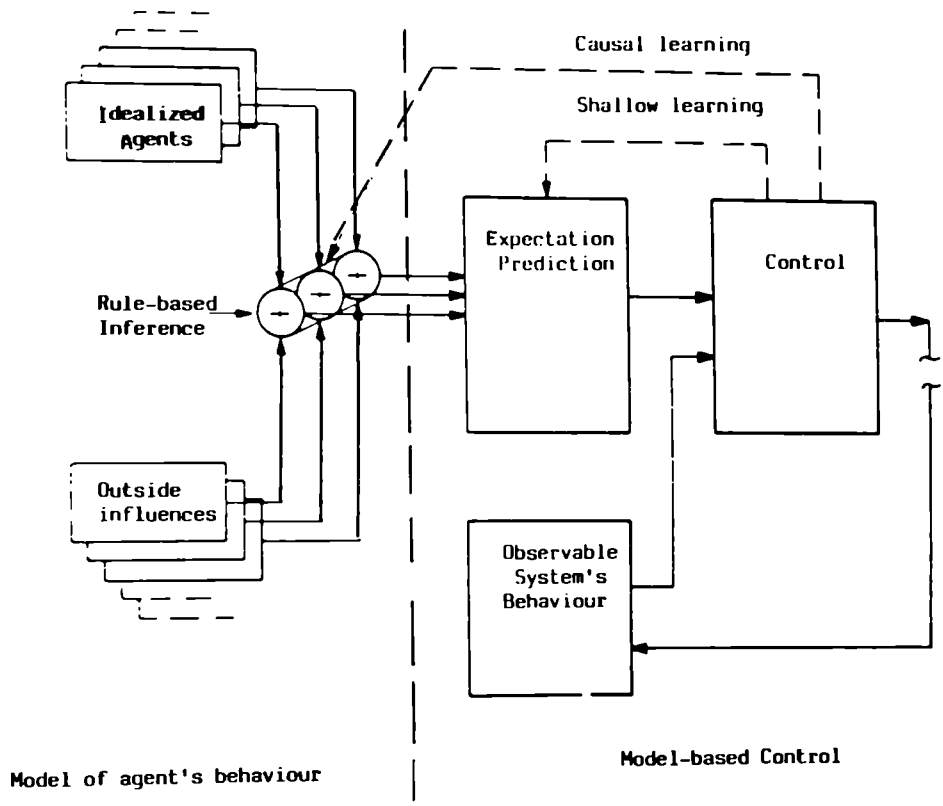
- heat pumps

- various individual consumers.

Consumer types that consist of a very large number of individual consumers, like storage heaters, are split into several groups which are treated independently by the distribution company. For each type of consumer or each group of consumers there exists an individual tariff contract, which can be quite complex. These contracts lay down conditions referring to switch-on and switch-off times and/or timespans, the number of times such consumers may be switched off, for how long it has to be switched on again after a switch-off, etc...



2-1: Load distribution over a cold winter day



3-1 : Concept of Model-based Control

3.1.] The control Paradigm

To avoid emergency situations and unwelcome surprises, short and long term planning of power demand (in most cases this actually refers to power production planning) is carried out.

The actual control of the distribution then follows this plan, and as long as nothing unforeseen is happening, the control itself is simple and straightforward.

Unfortunately, the planning itself is not that straightforward, as it is based on assumptions with varying and unknown degrees of uncertainty. Therefore, quantitative prediction is not possible, basically because of two reasons :

a - The outside influence as the weather cannot be predicted in a definite form
b - The behavior of consumers, i.e. their reaction to these influences cannot be predicted, as this would require a complete model of their behavior.

The behavior of consumers can only be described by a qualitative model. To construct such a model, a distinction is made between an idealized consumer - or more generally an idealized agent - and the outside influences. In these terms, behavior is understood as the reaction of an agent to an outside influence. Thus, to model the idealized agent's behavior means modelling his behavior without outside influences. Since this behavior is not known and in practice is not observable, this model must be based on assumptions.

Outside influences are observable incidents, so they can be considered as known but of course not necessarily as predictable. Furthermore what is not known and also influences and the idealized agent's behavior. The best way to express this seems to be by a rule based inference mechanism, the outcome of which could then be considered as the qualitative model of the behavior of the collection of all the agents. Figure 3-1 gives the conceptual overview of this approach.

This qualitative model of the agent's behavior would then lead to the expectation of the power demand for the near future and therefore form one of the two inputs for the control process. The other input is of course the observable system's behavior, which in this application is the actual load of the distribution network. Based on these two parameters the control decides if and how to react to either avoid overload or to establish more economical distribution.

The first steps towards constructing such a qualitative model have been taken by investigating some typical functions relating outside influences to consumer's behavior.

One of the main influences is without doubt the outdoors temperature with respect to the energy demand of private houses. In the context of electricity distribution, this of course relates only to houses using electrical means for heating, such as storage heaters, direct heaters, or heat pumps. Based on this functional model a first, simple qualitative model has been "handcrafted", not yet built on rule-based inference but on predetermined dependencies. Nevertheless, the current architecture and solftware development is targeted towards such a rule-based model.

This simple qualitative model provides one of the inputs for the system's expectation and prediction module, the other being "historical" information, i.e. available data about load demand of previous days and years for similar situations.

The expectation module is basically a planning module, which is based on the control objectives and the existing load-distribution. It takes into account the functional model of consumer behavior together with the outside influences and produces guidelines for the plant control together with a new load-distribution, constituting the expectation.

The control guidelines, which consist of a switch plan, together with the observable system's behavior, which in this application is the actual load of the distribution network, form the basis for the control during the day.

The output of this planner - the plan - contains a sequence of single switch action entries, each consisting of the time of day, when the action is to be performed, the name of the consumer group concerned, and the kind of switch (on or off).

The plan is used by the control component of the entire expert system, which besides supervising the network carries out the actions suggested by the plan, unless anything unexpected is being detected.

4] Planning

For the remainder of this chapter we will concentrate on the planning module and not so much on control, simulation, and real-time requirements, which are described elsewhere ([12], [13]).

Due to the first two objectives stated in part 3, the planning can be subdivided into two different subtasks. One is responsible for overload situations, i.e. has to plan the switching off of consumers, and the other is responsible for area, i.e. it has to plan the switching on of consumers. The third mentioned objective, namely to meet the various tariff conditions can be seen as an overall constraint for both planning tasks.

Irrespective of the specific objective of the two subtasks, the general concept that both follow can be described by the following steps :

a - Find a "critical" situation. This is not done in a time sequential fashion, starting in the morning and proceeding to the end of the day, but by looking at the overall states of the expected load and then focusing directly on those areas where such situations can be expected, employing a multi-planning strategy in terms of [14]. What "critical" means is determined by the objective of the relevant module, either referring to a possible overload or to an undesirable underload.

b - Select a corrective action and assess the effect of that action.

c - Find the next "critical" situation.

Since each planned action changes the overall situation, the subsequent situation selection has again to be approached from an overall viewpoint. Furthermore, each planned action constitutes a constraint on the further planning. Since such planning strategies, due to the various constraints, might easily lead into blind alleys, effective means of backtracking have to be established (see part 4.2.).

4.1.] Planning Strategies

Two modules for planning have been developed :

. The Switch Off Planning (SFP). This module selects and orders actions in order to avoid overload.

. The Switch On Planning (SNP). This module selects and orders actions in order to avoid underloads and to satisfy the minimum supply times in accordance with the tariff rules.

Both modules follow the basic stragegy outlined above. But the first question clearly is, which to start first ? This question concerns the overall strategy and is independent of the internal strategies of the two modules. As will be shown later, this distinction is clearly reflected in the architecture of the system. Currently there are two different strategies for applying the two modules to a given problem situation. They are :

Strategy 1 :
Apply SNP first, as this deals with tariff constraints that have to be followed anyway. Since it tries to shift load into low-demand areas, it is quite likely that it reduces some load in the peak areas.
Then apply SFP to concentrate on any remaining critical areas.

Strategy 2 :
Apply SFP first if substantial overloads are to be expected. Reduce only the most critical ones, not necessarily ending with a "clean" situation.
Now apply SNP as above.
Apply SFP for any remaining overloads, if some remain, otherwise terminate.

The decision on which of these two strategies to follow is dependent on the actual situation. The overall strategy module contains rules that describe the condition for each strategy and the system monitors the application of the strategy.

4.1.1.] SFP Module

The main difficulty lies in selecting an action. An action is a tuple consisting of an interval and the name of a consumer that has to be either switched on or off during this interval. Selecting such an action really means achieving two subgoals simultaneously, taking into account that choosing an interval and selecting an appropriate consumer for this interval are not independant tasks.

The first step in selecting an interval is done in an overall way, based on such criteria as :

- length of an interval of likely over - or underload

- absolute height of such interval

- area of that interval (integration over time).

Once such an interval has been established, a suitable consumer must be found. First, a candidate list of possible consumers is drawn up. All candidates are constrained by their tariff rules, which, for example, determine the maximum timespan for switch-off periods. If these allowable intervals are shorter than the selected one, the full interval has to be split into subintervals. So this step ends with a set of suitable consumers with overlapping subintervals.

Some higher level strategies or heuristics then decide the priority ordering of these actions - consumers are weighted. Such heuristics take into account the type of consumer (industrial consumers usually have the lowest priority for a switch-off actions), the time of day or weather conditions. As soon as an action has been definitely selected, its affect on the overall situation is checked by simulation. If it is no positive, this action will be rejected. Depending on the actual case, this might lead to abandoning the current interval and looking for another, coming back to the first afterwards. This is done by means of dependency-directed backtracking.

4.1.2.] SNP Module

The switch-on planning has to schedule the consumption periods of those consumers, that are only to be switched on for certain parts of the day according to their tariff contracts. Thereby it has to "fill up the troughs" of the overall power consumption curve.

In order to achieve its goals the SNP requires a slightly more complicated control strategy than the SFP. This refers to the control cell built to overlay the basic switch-on planning cycle as described above. The strategy pursued by this control procedure can be divided into the following three steps :

- First choose those consumers, that have to be switched on anyway during one or several fixed time intervals. For example some groups of storage heaters must be switched on exactly between 10 o'clock in the evening and 6 o'clock in the morning.

- Secondly satisfy the minimum switch-on period constraints. At this step the switch-on planning cycle is run. However, the final goal is only to plan actions to make sure that a consumer that has to be switched on between N and N+K hours will be on at least N hours. It may be, that this step ends without having achieved its goal, the reason being that there are no more intervals during which the consumption curve is sufficiently below the limit-curve, while there are still some consumers for which the minimum switch-on is not covered by the plan. In this case the control restarts this step giving it a new temporary limit curve that has been slightly augmented in comparison to the original one.

- Thirdly, satisfy the optional switch-on period constraints. Here, the proper switch-on planning loop is started again, now having the goal of trying to satisfy the amounts of time some consumers can be switched on but do not have to be. In other words, for a consumer, that has to be switched on between N and N+K hours, this step tries to establish a suitable interval for the remaining and optional K hours. This step will end when there are no more time intervals during which the required actions can be taken without exceeding the overall limit.

If the control had to perform a restart of step 2 after a temporary limit increase raising, one consequence would be that some planned actions of the SNP will cause exceeding the (original, still valid) limit curve. This will be reported by the control to its superior, overall control level.

4.2.] Backtracking

4.2.1.] General

An important component of the planner is the backtracking mechanism. Backtracking gives the planner the flexibility that is essential for performing load control over the power distribution network. Of course, one of the advantages system is the possibility of backtracking actions, together with their supposed effect on the network, since the actions are only virtual until their really executed. Contrast this to the performance of actions directly after their selection without the use of any plan. In the latter case, an action that places the network in an unsatisfactory state can be revised, but it might not be possible to correct the impact it had on the system during the time between its execution and its revision or only by means of great efforts.

The backtracking facility of the planner in the load management system can incorporate two backtracking functions. Due to the events they are triggered by, they will be called "internal backtracking" and the "external backtracking".

4.2.2.] The Internal Backtracking

Internal backtracking is backtracking in the sense mentioned above. It is applied as the planner builds up the plan. If the plan is only partly created and the planner cannot find any possible way to go forward it, it has to go back. It has to remove one or several actions from the part of the plan already existing and try to find a better course to accomplish the plan.

To be able to perform backtracking in the indicated manner, it must be possible for the planner to recognize, that some obstacles to proceeding with the plan are caused by actions planned earlier. Furthermore, it must be possible to identify these actions, and realize how the consistency of the plan can be maintained when actions are withdrawn. Therefore, during the creation of the plan a dependency network is constructed, describing the causal relationships between the actions stored in the plan and the reasons for their selection as well as their assumed influences on the system being forecasted. Thus all information needed when backtracking can be readily obtained. The mechanism applied closely follows TMS-like solutions ([15], [16]).

4.2.3.] The External Backtracking

External backtracking of parts of the plan is initiated from "outside the planner". While the control is already executing the plan it may happen in two ways :

a - The control discovers that the system is about to enter a prohibited state, i.e. it has become very likely, that the power limit will be exceeded in the near future, although up to that moment the plan has been followed correctly. Such situations may arise since the plan is based on uncertain knowledge about the future and there always will be small deviations of the reality from the prediction. In this case, the plan has to be extended by adding one or more actions to it.

  The extension of the plan is triggered by the control, which at the same time chooses actions helping to prevent the emergency state of the system. Afterwards, the planner integrates these new actions into the plan, which basically means checking the consistency of the extended plan. The integration of given new actions into the plan may involve the revision of some other planned actions, due to conflict with the new ones. This handled by means of the same techniques as in internal backtracking.

b - This kind of backtracking is rather a replanning than a proper backtracking. It will be applied when the control gets information about unexpected events which the planner could not be aware of ; e.g. the outside temperature might drop unexpectedly. In this case the plan is based on at least partly wrong assumptions. Now it would not make sense to try to solve the problem by only modifying the plan slightly. Rather from a certain point in time t0 onward, a completely new plan must be created having all the actions in the plan executed before t0 as a constraint. At t0 the control switches over from the old plan to the new one.

Type a) of external backtracking is the preferred one, as it is accomplished faster and causes less time problems.

However it is not always an easy task to decide whether type a) - the plan correction - is really sufficient or whether type b) - the total replanning from a certain point of time on - is required. Hence the control relies on a rule-based knowledge source indicating in each case which type should be favored.

5] Implementation and Results

The planner together with the entire load management control system has been implemented on a Lisp-machine. This system controls a simulated power distribution network, which consists of numerous simulated consumers and is based on real data obtained from an existing power distribution network.

5.1] basic Tools

The basic tools of any knowledge-based system are a suitable represenation language and an efficient inference engine. Both have been jointly developed in the KRITIC project, especially oriented towards systems to be built for industrial applications.

The knowledge representation language called AVALON [17] has been designed to meet the special requirements of industrial applications. It is a frame-based language building on structured objects defined in a type-hierarchy.

Its main features are :

- It allows the creation of large knowledge bases. For example, the network representation for the distribution network currently contains over 15000 instances and is likely to increase to 50000 in real applications. To handle such large sets requires very efficient means.

- To speed up inference procedures working on this representation, AVALON allows partitioning of the knowledge base so that only a small part of it needs to be loaded at any one time.

- It not only allows inheritance of attributes but also of relations. Relations exist in a type hierarchy jus as objects do, and relation instances associate object instances with other object instances.

The inference engine, called MIKIC ([18]) is based on the object oriented paradigm. It allows forwards and backward chaining, definition of rulesets and the specification of various "evaluation styles" for such rule sets.

One of the more important features is that it has a compiler to increase the execution speed of the rule interpretation (in the prototype environment describes here, up to 300 rules per second).

The other important aspect of MIKIC is the handling of generic rules. Generic rules allow a very abstract way of formulating knowledge. They are connected to object-types. If such a generic rule is invoked, it will automatically be applied to all instances of this object. Furthermore it is inherited by lower levels of the object-type hierarchy. Thus extending the knowledge base by further objects or instances does not require any change in the generic rules. This is an important feature especially for industrial applications, since any plant generally consists of large numbers of "objects" or elements belonging to the same class. Instead of addressing them individually with a tremendous amount of (very similar) rules, only one generic rule is required.

5.2.] High-level Control

As noted in CHAPTER ONE, the structure of control seems to us an important subject as well as a major step in solving the control problem.

The approach cited in part 3 of the first chapter is called "Cell/Tissue".

Cell/Tissue comes in part from deep studies of the literature on blackboards as well as blackboard implementations and in part from techniques for control of large database system as well as operating system theories.

The Cell/Tissue approach ([19], [20], [30], [31]) is based on on defining independent control clusters, or "problem-solving islands" (to be compared with the works of Stefik [8], Minsky [21]. The structure of each cluster follows a three-level hierarchy of control objects for expressing its specific problem solving goals. The underlying claim is that these three levels are sufficient to express any well-defined task. In other words, each complex problem can be hierarchically structured so that on each level of the hierarchy such three level cluster can be defined.

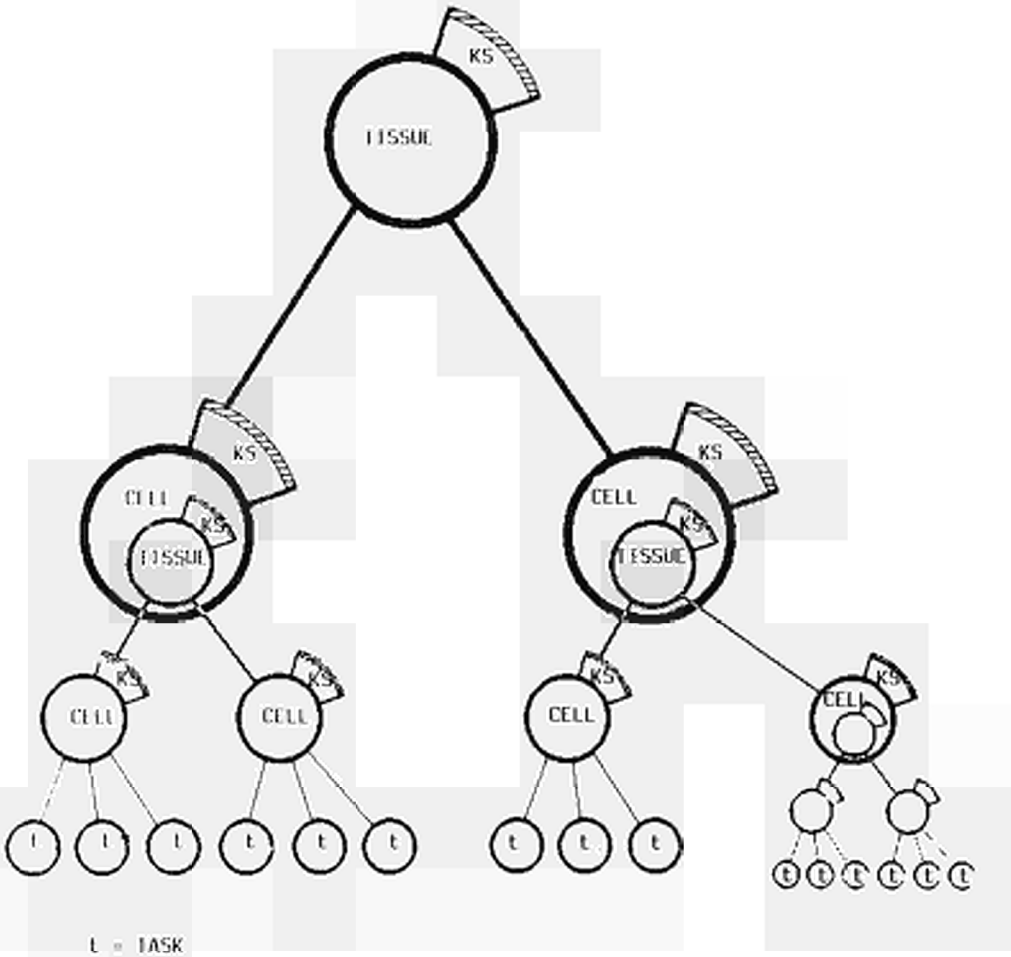
Basically, this means splitting up a problem according to different, ordered objectives and, on each level, concentrating only on this objective.

The objects of these control levels in Cell/Tissue are called tissue, cell, and task respectively. Basic actions to be executed on the lowest level are called subtasks. Figure 5.1 shows these three levels and how the clusters can be nested.

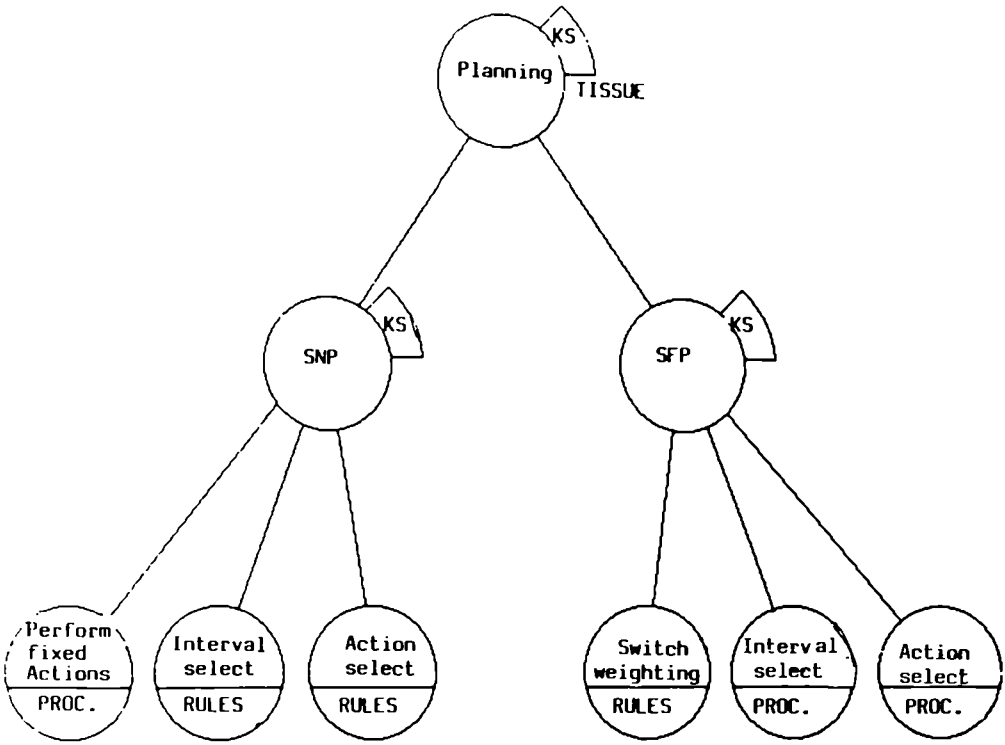
The two higher objects, tissue and cell, have local knowledge bases attached to them, which, similar to the agenda in BB system, determine the sequence of control over the lower objects.

a - The tissue
    The tissue is the highest control object, expressing the general problem solving strategy. In this sense, it is reponsible for planning the problem solving process. In the planning system of our application, the tissue represents the overall strategy, deciding in which order the SNP and SFP modules have to be scheduled.

b - The cell
    The cell is responsible for ordering the problem solving steps, once the strategy has been determined by the tissue. On this level, the overall problem is already decomposed to identifiable steps such as interval selection. The cell itself does not know how to achieve such a goal, but it identifies this goal as a task that can be scheduled.



5-1: Principal CELL/TISSUE structure



5-2: The Planning Control Cluster

c - The task
The task orders the final execution. It can be either described through rulesets, where each rule or rule subset, is considered a subtask that is simply run, or they can be procedures. To demonstrate the flexibility of this approach some tasks in the planning systems have been implemented as rules and some as procedures.

The important fact is that the way a specific task is implemented does not concern the next higher level.

On the cell level, it is irrelevant how the lower levels tasks are implemented. Likewise for a tissue it is irrelevant how a cell is implemented.

In fact as shown in fig 5.1, a cell might be represented by a complete control cluster itself. Depending on the viewpoint this cluster would be either seen as a cell - from its own tissue - or as a tissue - from within this control cluster.

Internally, each tissue and cell consist of a number of actions called either tissue actions or cell actions. They can be hierarchically ordered but their scope and their results are confined to the corresponding control levels.

As will be shown later, the planning tissue has to carry out various actions such as initialisation, dealing with basic certainty functions, resetting planned curves etc., before it passes control to the lower cell-level.

The planning system has been implemented in one control cluster as shown in figure 5.2. The generic control structure of the planner is described in figure 5.3.

The types of the control objects on the three control levels are tissue, cell and task. The relations between the drawn objects are e-R (eval rule), r-a (run action, for starting submodules of cell or tissues, so called cell and tissue actions), s-c (start cell), r-t (run task), and r-st (run subtask).

The generic control hierarchy of the planner is described in figure 5.4.
The root PLAN-CONTROL of this tree is a tissue, using some rules for performing control over the whole planner. One recognises the two cells SNP-CONTROL and SFP-CONTROL, started by tissue action ACTIVATE-PLAN via s-c and each controlling one component of the planner. The name of the relations are described at figure 5.3.

As one can see from figures 5.5 and 5.6, the SFP control and SNP control are differently implemented : the task TASK-SFP-WEIGHTING uses a rule set, as do APPLY-SNP-ACTION or OBSERVE-SITUATION (SNP module), while the remaining tasks are encoded as procedures.

6] Links with Chapter 1

As we have seen, it is essential in KBS applications for industrial control to define, prior to any implementation, a taxonomy of the problems to solve.

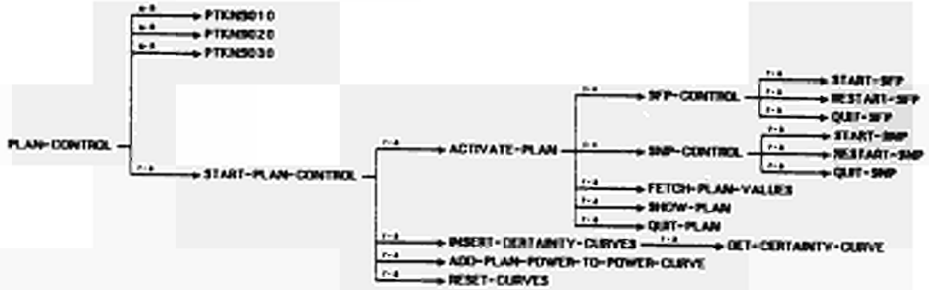
In the application described, this taxonomy covers the following particular topics :

- planning or Power-load-control

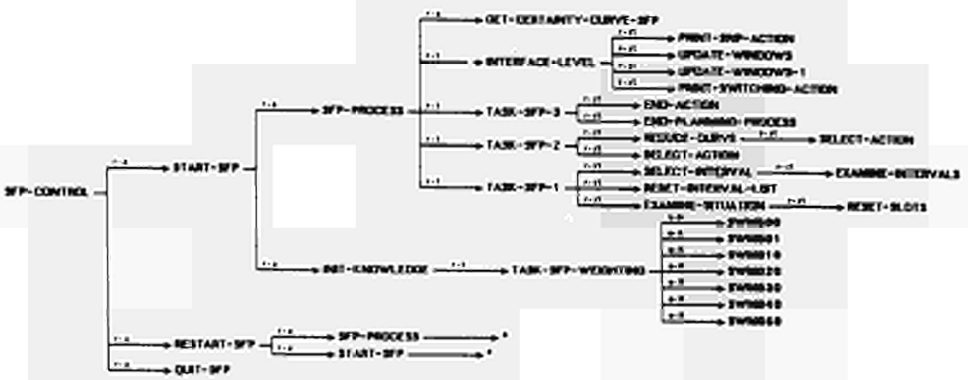
- external backtracking

- SFP or SNP planning

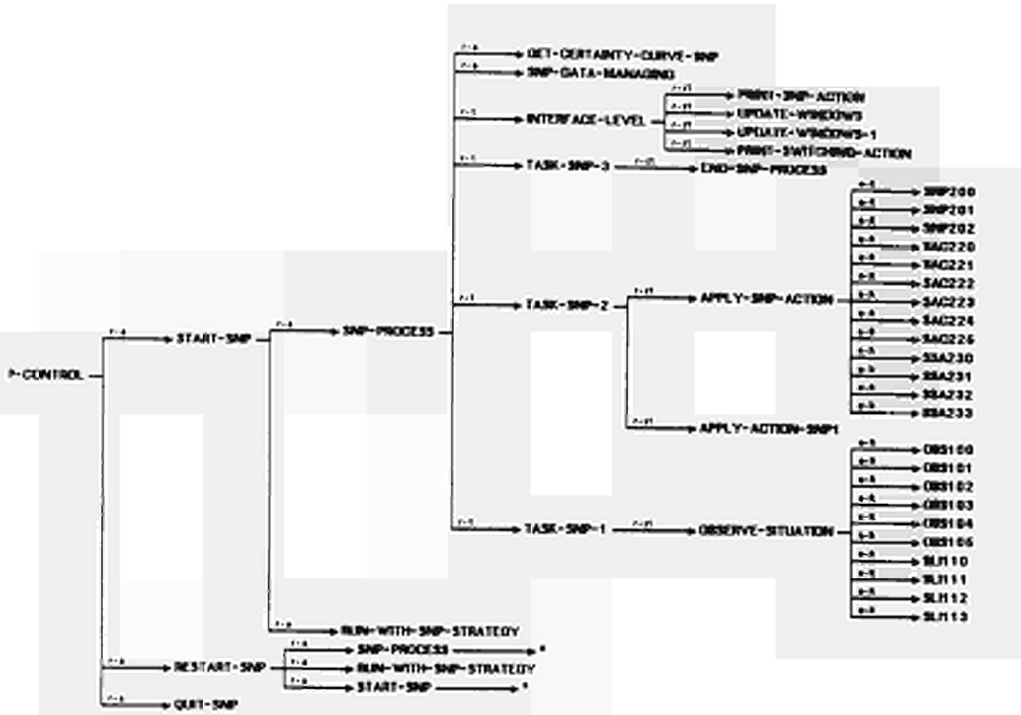
5-3: Generic Structure



5-4: General Control Hierarchy



5-5: SFP Control



5-6: SNP Control

- internal backtracking

- interval selection, consumer selection, action selection.

In fact, basically three decision problems have to be solved :

1 - Decide when, and how to start planning or power-load-control

2 - Decide how to accomplish power-load-control

3 - Decide how to do planning.

In the Cell/Tissue framework, these decision problems correspond to three different hierarchical clusters, each represented by a Tissue-Cell-Task control structure. Problem 1 is essentially a problem of control over the decision-problems 2 and 3 which are themselves independent, while problem 1 is not concerned with the specific ways problems 2 and 3 are solved.

The corresponding modules, or problem solving islands, generates EVENTS that are used by the top-most control level to generate a plan : run the planning module, then run the power-load-control module. This will generate other EVENTS such as "a discrepancy is observed", which in turn will motivate another decision, namely to re-plan, to wait, or any combination.

The first analysis to perform in order to be able to implement the KBS control is thus to :

- IDENTIFY the problem-solving clusters

- IDENTIFY the EVENTS that need to be controlled

- USE these clusters and events to define the basic control loops or mechanisms needed.

To accomplish this, the works of Lansky [10] on specification language for tasks synchronisation seems to be very relevant. A similar work, although with major technical differences, but conceptually near this approach, is the one by D'Ambrosio-Fehling-Forrest-Raufels-Wilber [22].

A very interesting discussion on Planning probles which addresses problems that are essential but not talked about in this paper is the R.E. Korf paper [32].

When the problems to solve have been well defined, one has to go further in the analysis by specifying the kind of control is needed to implement the correct problem solving methods.

Basically, what the control, here, has to perform at this level is :

a - Choose a subproblem solving island if needed (the Tissue level) ; in the previous example, "planning or power-load-control" for the top-most cluster, or "SNP-plan, SFP-plan" for the planning cluster ;

b - Inside a subproblem (the Cell level) decide on the sequences of solving tasks to be achieved, in the previous example "select interval, select consumer, select final action" ;

c - Inside an elected task (the Task level), decide on the basic ordering of final computing actions (the Subtask level).

This hierarchy is reflected through the Cell/Tissue structure and specified by using the general framework of problem solving islands and events previously

explained.

FURTHER DISCUSSION AND CONCLUSION

The mechanisms involved in "replanning" or constraints problems introduce difficult challenges in the development of KBS for industrial process control, to understand them, one has to think in terms of classical Optimal control theory (Belmann, Pontryaguin [23] [24], as well as more advanced studies such as viability and cognitive process [Aubin, Changeux [25] [26].

The goal of Optimal control is to optimise a criterion (such as power load) by finding a control function (or map) over a dynamic system.

In fact, this system is parametrized by a control space, where the control map is defined, and allows building a constraint space identical to the one introduced in non-dynamic optimization.

The discretized version of this problem is exactly to define an optimum sequence of control actions (a plan) as the solutions of an optimization problem under evolving constraints.

Viability theory precisely studies the dynamic evolution of the constraints set defined by the dynamics, which represent the model of the system to control. When that model is well-defined and satisfies mathematical assumptions, the theory gives algorithms to build the optimal plan. In such cases, the "replanning" problem is not relevant, except if the dynamic model used is not good enough. Part of the algorithms used are based on constraint satisfaction.

In our case, either due to the complexity of the process or its size, full mathematical models are out of reach. They are replaced by the qualitative, or experimental representations previously mentioned. In fact the rules used to achieve a plan can be seen as a synthetic representation of the analytical algorithms, or as "shortcuts" in the incremental steps of such methods. But since the representations cannot be seen as a well-defined exact model, the constraint sets could suffer from incompletude or be ill-defined.

Evolving in a logical framework, the qualitative representations, in the face of these problems, can only be ckecked by coherency/completeness methods in their reasoning environment models.

In case of discrepancies, the plan can only be adapted by going back into its creation (the various planning islands involved) to incorporate the relevant observations and replan. The planning process cannot be self-adaptative unlike the case in many well-defined optimal control problems.

That is why the mechanisms involved in constraints problems and replanning are as described below.

- For the "replanner control" the main problem is with the backtracking decisions, it could involve a complex process such as dependency directed backtracking (see de Kleer, Mc Allester, Forbus [27] [28] [29]) possibly essentially defined by algorithms with few heuristics.

- The detection of constraints problems (incompletude or need for relaxation) is a mixed mechanism. Basically it is relevant to the de Kleer ATMS (see [16]) for the first problem, and to classical TMS for the second one. However, once the detection is made, the viewpoints to propose to the planning islands cannot be built other than by using strong heuristics.

To conclude, we believe that this work shows the feasibility of a KBS for real time process control of an industrial system. Moreover, it also demonstrates the role that KBS can, and perhaps should, play when the mathematical modelling, although conceptually essential, comes up against the difficult problem of effectively solving in real time a large-scale industrial control application.

BIBLIOGRAPHY

[1] G. Williamson - J. Butler - E. Gaussens - S. King - V. Khong
"Using a KBS in Telecommunications"
Proc. Esprit Conference 1986 - North Holland - 1986

[2] B. Hayes-Roth
"A Blackboard architecture for control"
AI Magazine 26, pp 251-321, 1985

[3] J. de Kleer - J. Doyle - G.L. Steele - G.J. Sussman
"Explicit control of reasoning"
AI Memo 427, June 1977

[4] P.J. Hayes
"In defense of Logic"
IJCAI-77, pp 559-565, 1977

[5] M.R. Genesereth - D.E. Smith
"Meta-level architectures"
Stanford Heuristic Programming Proj., Draft - Memo HPP-81-6 - 1981

[6] D.B. Lenat
"BEINGS : Knowledge as interacting experts"
IJCAI-75, pp 126-133, 1975

D.B. Lenat - F. Hayes-Roth - P. Klahr
"Cognitive economy in Artificial Intelligence systems"
IJCAI-79, pp 531-536, 1979

[7] E.D. Sacerdoti
"A structure for plans and behavior"
Elsevier-North Holland New York, 1977

[8] M. Stefik
"Planning and Meta-Planning (MOLGEN : Part 2)"
Readings in AI Webber & Nilsson ed. TIOGA pub., pp 272-286, 1981

[9] J.J. Anton - E.A. Feigenbaum - H.P. Nii - A.J. Rockmore
"Signal-to-symbol transformation : HASP/SIAP case study"
AI Magazine Spring 1982, pp 23-35, 1982

[10] A.L. Lansky - S.S. Owicky
"GEM : a tool for concurrency specification and varification"
Proceedings of the 2nd annual ACM Symp. on Principles of distributed computing, pp 198-212, 1983

[11] M. Georgeff
"The representation of events in multi-agents domain"
Proceedings AAAI 86 (Science), pp 70-75, 1986

[12] H.W. Fruechtenicht - T. Wittig
"Ein ansatz fur echtzeit - Expertensysteme (an Approach to real-time E.S.)"
INTERKAMA '86, Springer Verlag, 1986

[13] T. Wittig
"Power distribution falls under KRITIC's eye"
Modern Power System, January, London, 1987

[14] J. Hertzberg
"Plannerstellungs-methoden der KI"
Informatik Spektrum, pp 149-161, 1986

[15] J. Doyle
"A Truth Maintenance System"
Artificial Intelligence (12), pp 231-272, 1979

[16] J. de Kleer
"An assumption based truth maintenance system"
Artificial Intelligence, pp 127-224, 1986

[17] S. Varey
"AVALON"
Technical memo nb 8, Esprit project 387, Krupp Atlas Elektronik - Bremen,
1987 (forthcoming)

[18] V.H. Khong
"A study of rule-based, frame-based and constraint representation in AI"
Phd Thesis, University of London, 1985

[19] R. Davis - R.G. Smith :
"Negotiation as a metaphor for distributed Problem solving"
Artificial Intelligence 20 (1), pp 63-109, 1983

[20] E. Gaussens et al
"Cell/Tissue"
Technical memo nb7, Esprit project 387, Krupp Atlas Elektronik - Bremen,
1987 (forthcoming)

[21] M.A. Minsky
"A framework for representing knowledge"
In P. Winston (Ed.) The psychology of computer vision, Mac Graw Hill, 197

"Steps toward artificial intelligence"
In E.A. Feigenbaum & J. Feldman (eds.), Computers and Throughts, Mc Graw
Hill, 1963

[22] B. D'Ambrosio - M.R. Fehling - S. Forrest - P. Raulefs - B.M. Wilber
"Real time process management for materials composition in chemical
manufacturing"
IEEE EXPERT Summer 1987, pp 80-93, 1987

[23] R. Bellman
"Introduction to the mathematical theory of control processes"
Academic Press, 1967

[24] L.S.Pontryaguin - V.G. Boltiansky - R.V. Gamkrelidze - E.F. Mischenko
"Mathematical theory of optimal processes"
Interscience, 1962

[25] J.P. Aubin - H. Frankowska
"Viability and control of systems with feedbacks"
Personal communication, CEREMADE, Universite Paris IX Dauphine
Place du Marechal de Lattre de Tassigny, 75116 PARIS, 1986

[26] J.P. Changeux - P. Courreges - A. Danchin
"A theory of epigenesis of neuronal networks by selective stabilisation of synapses"
Proc. National Academy of SCIENCES OF usa, 70-2974, 1973

[27] D.A.Mc Allester

"An outlook on Truth Maintenance"
MIT AI Memo Nb 551, August 1980

[28] D.A. Mc Allester
"The use of equality in deduction and knowledge representation"
MIT AI-TR550, January 1980

[29] K. Forbus - J. de Kleer
Tutorial AAAI 1986

[30] D. Corkhill - K. Gallagher - K. Murray
"GBB : A Generic Blackboard Development System"
Proceedings AAAI 86 Engineering, pp 1008-1014, 1986

[31] E. Durfee - V. Lesser
"Incremental Planning to Control a Blackboard-based Problem Solver"
Proceedings AAAI 86 Science, pp 58-64, 1986

[32] R.E. Korf
"Planning as search : a quantitative approach"
Artificial Intelligence (33) September, pp 65-88, 1987

Project No. 387

USING KBS IN TELECOMMUNICATIONS 2

Authors: George I. Williamson *
John Bigham +
John W. Butler *
Simon G. King *

1. INTRODUCTION

1.1. The Scope of this Paper - The Telecommunications Domain

This paper describes primarily the work leading to a demonstrator Knowledge Based System (KBS). The KBS is targetted at operation and maintenance of advanced telecommunications switching systems. The work described here builds on that discussed in [17].

The KRITIC project as a whole has broader objectives than that of operation and maintenance of telecommunications switching systems. Therefore, in order to place this work in context, a brief overview of the project is given below. The results obtained in other work areas within the KRITIC project, for example load control of electricity distribution networks (Wittig [18]), will be and have been described separately.

1.2. Overview of Project 387 - KRITIC

The overall objective of the KRITIC project is to make possible the development of KBS for complex industrial application areas. The collaborators in this project are: Krupp Atlas Elektronik - prime contractor (West Germany), British Telecom (U.K.), Framentec (France) and Queen Mary College (U.K.).

The KRITIC project addresses the task of constructing KBS for use in industrial control application areas. It has been argued that these applications define a class of KBS not yet researched sufficiently well. The major attributes of this class of KBS are:

- high levels of organisational complexity in an engineering domain
- large numbers of input and output interactions
- time dependent reasoning
- learning/adaptation

The project extends over 3 years and will absorb 36 man-years of effort amongst the four collaborators.

The project plan has two main stages. The first is intended to produce a set of tools and facilities constituting a KBS development environment. The second involves the use of the development environment to produce two

---

* British Telecommunications plc, TA12.2.2, 151 Gower Street, LONDON WC1E 6BA, U.K.
+ Dept. of Electrical and Electronic Engineering, Queen Mary College, Univ. of London, Mile End Road, LONDON E1 4NS, U.K.

significant KBS representative of this class of KBS. One of these is targetted at operation and maintenance of telecommunications switching systems and the other at load control of an electricity distribution network.

The project started in February 1985 and the major project milestones are listed below:

PHASE	DATE	DESCRIPTION
Phase 1	June 1985	Literature Survey
Phase 2	December 1985	Initial Research
Phase 3	July 1986	Two Prototype KBS (Domains 1 and 2)
Phase 3	March 1987	KBS Development Environment
Phase 4	December 1987	Two Full Specification KBS (Domains 1 and 2)

The project is currently divided into three work areas: Domain 1 which is principally interested in the application of KBS to telecommunications switch operation and maintenance; Domain 2 which is principally concerned with KBS in control of electricity distribution networks; and, Kernel which takes a more top down approach to investigate domain specific methodologies and to generalise domain independent approaches from them. Domains 1 and 2 are staffed by British Telecom and Krupp Atlas Elektronik respectively. The Kernel work area is staffed by Queen Mary College and Framentec.

2. TELECOMMUNICATIONS DOMAIN REQUIREMENTS AND OUTLINE OF PAPER

The tools produced in the KRITIC project are intended to be generally applicable to control problems in industrial systems. A more general description of the aims of the KRITIC project is given in [8]. This paper focusses on recent experience gained using tools appropriate to the telecommunications switching domain. In particular, it focusses on experience gained with tools applied to the problem of maintenance of a telecommunications switching system.

2.1. Background

In the broadest definition the telecommunications switching domain could include maintenance, operation and aspects of design of advanced digital switching systems. In these systems maintenance activities include: diagnosis of faults to replaceable units (usually printed circuit board level); processor maintenance (reloads, patches etc); and, exchange line and circuit testing. Operations activities include: billing and accounting; analysis of exchange statistics; provision of subscribers facilities and circuits; and, network administration. Design of telecommunications switches is a more ambitious target than either operation or maintenance; however the KBS development environment tools have been constructed bearing in mind their applicability to design for operation, maintenance and testability. The particular sub-domain chosen for the project work to date has been that of maintenance of the Monarch IT440 PABX.

The use of KBS in telecommunications switching domains is increasing. The need for KBS technology in the field has been described by other workers. Seviora [14] stated that the ultimate solution to the problem of field debugging of current generation telecommunications switching systems may come from the introduction of KBS debug systems. Goyal et al [6] describe a expert system for maintenance of an earlier generation telecommunications switch. In an earlier paper [17] we described the architectural requirements and results from construction of a prototype KBS.

2.2. Outline of Paper

This paper describes progress and results achieved in the application of KBS technology to the problem of telecommunications switch maintenance as a first step toward the more ambitious overall exchange management task.

The ways in which a layered blackboard architecture has been applied to the problem are described. The blackboard system has been integrated with inference mechanisms which allow forward and backward chaining and truth maintenance capability. A motivation for this approach has been the desire to accomodate flexible input of test and exchange status information relevant to ongoing diagnostic threads. Such information may include aspects of non-monotonicity. Further motivations are: the recognition of and the ability to cater for inconsistencies which arise out of the invalidity of underlying assumptions or incorrect user input; and, provision of greater flexibility and structure in inference control.

The blackboard system has been linked to a model of the exchange based on a knowledge representation language. The model comprises a number of distinct views of the exchange. These are: the physical units of the exchange (printed circuit boards, shelves etc.); the functional blocks, which map onto the physical units at various levels of detail; and, views which model the control behaviour of the exchange.

Furthermore, recent work, based on models of the exchange, has shown how the models may be used to provide examples for rule induction. Such techniques are expected to greatly ease the acquisition of domain knowledge.

2.3. Basic AI Techniques Employed in Problem Solution

The work described in [17] used a limited repertoire of basic AI problem solving techniques: for example, search, forward/backward chaining and frame representation schemes. The more recent work to be described here has introduced non-monotonic reasoning and truth maintenance, more flexible blackboard control and a more generally applicable knowledge representation language. In addition, an induction algorithm similar to ID3 has been applied to domain data in order to ease the knowledge acquisition bottleneck.

2.3.1. Basic Inferencing

Last year we described [17] how a simple forward/backward chaining rule based inferencing system called MIKIC stemming from work by Khong [8] had been applied to the Monarch PABX diagnosis problem. This system was well integrated with the object oriented base language [16] facilitating structuring of the rule base and integrating MIKIC with other tools. Indeed the object oriented aspects of MIKIC have eased its integration with a blackboard system called BBF.

2.3.2. Blackboard Architecture and Control

BBF is a blackboard system constructed for the project influenced by the work of Nii [11],[12] and Hayes-Roth [7]. MIKIC rulesets may form the knowledge sources associated with BBF. Aspects of BBF control have also been linked to a truth maintenance system (TMS) based on the work of McAllester [10].

2.3.3. More Advanced Inferencing

The TMS is logic based and is described in Bigham et al [2]. With this TMS, MIKIC rules may be represented as a network of nodes (propositions) and clauses (the relationships between nodes). The nodes have associated truth values which may be true, false or unknown. The network serves as a memory of partially computed results. The TMS will automatically propagate truth values through the network and identify any contradictions found. The TMS has been integrated with MIKIC providing a capability known as MIKIC/TMS. The TMS has also been linked to the management of hypotheses at one layer of the blackboard. TMS has also been used to perform state change propagation through a dependency network of exchange components.

MIKIC/TMS provides an integrated inferencing system which retains the basic simplicity and structuring of the MIKIC system but with the additional flexibility and power of the TMS. When a MIKIC backward chaining rule set is instantiated a TMS proposition network is automatically created. As the MIKIC control fires rules, truth values in the TMS network are updated and their consequences propagated. In operation the user may volunteer information at any time rather than waiting for MIKIC to ask a question. Furthermore truth values may be retracted at any time and any contradictions will be exposed.

2.3.4. Knowledge Representation

In the prototype KBS described in 1986 [17] the inferencing parts of the system were written so as to be generally applicable to types rather than instances. Details about the particular configuration of an exchange instance were recovered from a frame-like representation scheme called BALDRIC; more recent work has used a representation language called AVALON [20].

Such frame systems have a natural role in representing the relatively well-defined structural and functional information about the domain. Representations of functional blocks and physical parts of the system controlled are required at multiple levels of detail and generality. Representations are required which are able to handle the various replicated elements of the systems controlled. A number of discrete "views" of the exchange system controlled are also required including, the physical units, the functional blocks and models of the behaviour of the system under fault conditions. These views may be combined in a single model using a generalised object based representation system, such as AVALON [20] (with BALDRIC like capabilities added as domain specific enhancements).

2.3.5. Learning

The exchange models constructed using knowledge representation schemes such as AVALON and BALDRIC may be used in simulation mode to generate examples which link fault 'events' to symptoms. These have been used to generate decision trees using algorithms similar to ID3 [13]. Such rules are generated off-line. This approach to automation of knowledge acquisition has already been described by Bratko [3] in the medical domain. Decision trees

thus generated may then be used to form parts of knowledge sources in the overall blackboard architecture. So far this approach has been applied to simulations of two kinds, firstly to functional models of the exchange where the protocols of the exchange built-in diagnostic tests are encoded and secondly to dependency network models where models of the control and power dependencies have been encoded as a directed graph.

3. OVERALL DOMAIN ARCHITECTURE

The overall KBS architecture is based on a layered blackboard system, these layers correspond to abstract steps in the overall diagnostic problem solving strategy. The blackboard system also permits the increased modularity required by a large KBS.

Each layer on the blackboard system has a number of knowledge sources (KS) associated with it and each of these may access detailed information concerning the configuration of a particular exchange. Such a knowledge representation is encoded using the AVALON language. The overall architecture of the system is given in figure 1.

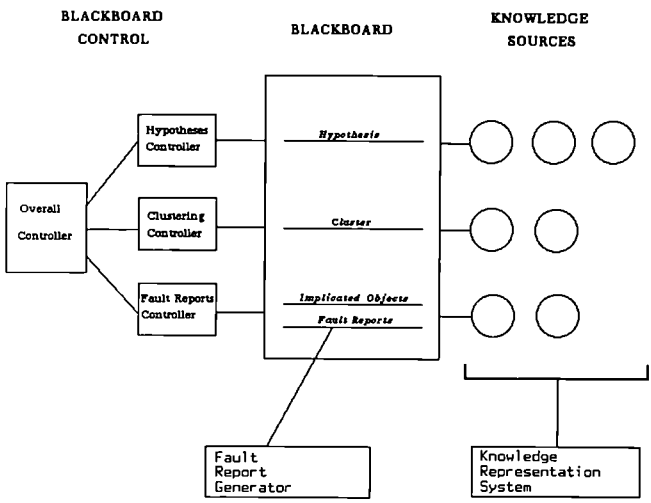


FIGURE 1 Overall Architecture

When faults occur in the Monarch PABX (and this is typical of other switching systems) fault symptoms are presented to the user by the built in diagnostic capabilities of the exchange. When they have been presented the general aim is to generate a minimal number of hypotheses corresponding to possible fault causes which may then be checked either sequentially or (in future developments) in parallel. It is desirable to allow flexible ordering of the hypotheses both to satisfy the users requirements and to permit adaptation mechanisms. In addition it may be necessary to integrate new events into existing diagnostic sequences and to take account of that new information accordingly. A layered blackboard structure has been devised to allow this kind of functionality, this is shown in figure 2 below.

3.1. Blackboard Layer Structure

Within this structure there are a number of steps.

1) Fault reports from various sources (e.g. diagnostic tests and lamp conditions) are translated to their implicated objects (i.e. the objects suspected as faulty).
2) These implicated objects are then clustered (since numbers of symptoms may be related to the same cause).
3) Then a number of separate hypotheses (possible faults and the maintenance procedures required to verify and repair them) are generated (for each cluster) and executed.

In normal operation the inference control has been constructed so that data will percolate up the blackboard through the layers until a solution is found. These steps correspond to the layers on the blackboard and to major steps in the general problem solving strategy. KS are implemented as MIKIC rulesets or as Lisp procedures and are allocated to each layer of the blackboard.

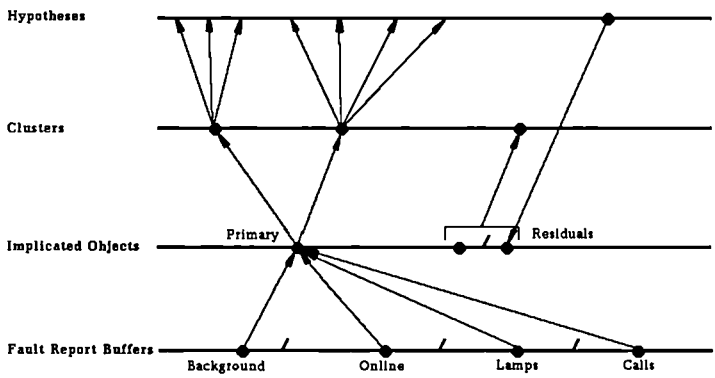


FIGURE 2 Blackboard Layer Structure

The general approach to KS control and execution uses the notion of KS priority. The blackboard has two control levels, one for overall control and another for control of the individual layers of the blackboard (see figure 3). There is now a much clearer separation of the control from the KS (inferencing) level than was present in the KBS described in [17].

Each layer controller has its own agenda, current list of KS activation records (KSAR), and list of associated KSs. The top layer has some methods for ordering the running of the layer controllers. The controllers are organised in a Flavor [16] hierarchy so that appropriate methods and instance variables are inherited by controllers from a prototypical controller. Such structuring of the control limits search by focussing at the layer and its associated knowledge sources. Furthermore, it allows greater flexibility in control by permitting different strategies to be implemented on each layer if necessary. For example at the clustering layer it is appropriate to execute all triggerable knowledge sources in one cycle and to merge results, whereas at the hypothesis layer only the first knowledge source is executed in one cycle.

The way that a layer on the blackboard works is basically this. At each cycle pre-conditions for each of the knowledge sources at a layer will be matched to the blackboard contents at that layer. If there is a match a KSAR is created. Once all KSAR's are created the list of KSAR's at a layer is sorted according to priorities at that layer. This sorted list forms the agenda at that layer. The KS are run in agenda order as controlled by the general scheduler. Generally KS at lower layers of the blackboard will have higher priorities. This strategy permits opportunistic scheduling of the KS. Future work may allow quasi-concurrent working of non interacting knowledge sources, and background processing of knowledge sources which require no dialogue with the user.

3.2. Examples of Knowledge Sources

The Monarch rulesets for the 18 month prototype [17] were used as the starting point for the blackboard development. However each diagnosis and therapy KS from [17] has been replaced by a number of hypothesis/test KS's (hypothetical fault events and checking procedures). There is now greater flexibility in control of the individual KS than was available previously.

Each distinct hypothesis/test may be executed as a separate backward chaining knowledge source. By way of an example some of the KS (implemented as MIKIC rulesets) at the hypothesis level are as follows.

Area-Level	Fault	Hypothesis KS
CSI-CARDS -	Powering	Hypothesis-1
	CSI card	Hypothesis-2
	Line Cards of CSI	Hypothesis-3
CSI-LINKS -	CSI sig/speech	Hypothesis-4
	CSI Edge Connectors	Hypothesis-5
	Line Unit Edge Connectors	Hypothesis-6
Line-Shelf -	Backplane	Hypothesis-7
	Piecewise Card Replacement	Hypothesis-8

The layer control uses information to allow it to select the most appropriate hypothesis. So that each hypothesis/test will have additional information to aid control and scheduling e.g.

```
scheduling information
     level (subsystem, system etc)
     likelihood - from fault report info.
     manpower cost
     cost in time to replace
     spares availability
     cost - locality to test desk
control information
     triggerability marker
     identified-fault marker
```

So far, these are preset but in future these additional items of information may be updated dynamically as the diagnosis progresses. The details of this mechanism are currently under review. Research is in progress to identify the best mechanism with which to take advantage of new information volunteered via the TMS capability.

At the hypothesis layer of the blackboard the knowledge sources implemented in MIKIC rules have been integrated with the TMS (see section 4). The proposition nets used by the TMS are set up as hypothesis rule-sets are generated (or invoked). These are effectively identical to the proposition networks corresponding to concatenated backward chaining rulesets with separate hypotheses OR'ed (or XOR'd) (see figure 3). For each hypothesis in a cluster a root hypothesis TMS node is automatically created by the blackboard control. Root hypothesis nodes in a cluster are linked by TMS clauses so that the hypotheses are 'ORed'. As knowledge sources are activated the TMS networks created by MIKIC/TMS for the backward chaining rulesets (hypothesis KSs) are linked to the hypothesis root nodes

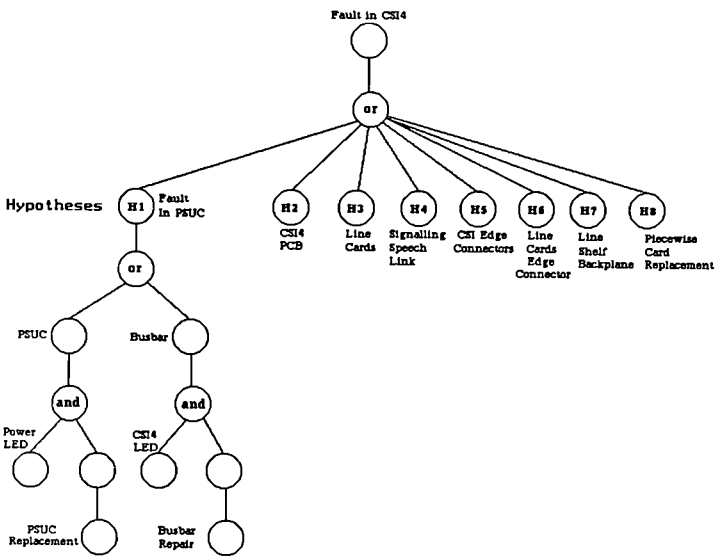


FIGURE 3 TMS OR'ed Hypothesis Tree

in the cluster. Once the network has been created, the user of the system may either answer questions as directed by MIKIC or volunteer information concerning leaf node states at any time via a menu. The control will react appropriately to this information as it is input. This allows much more flexible handling of the user dialogues at the hypothesis layer of the blackboard. If any nodes in the TMS are common between knowledge sources (either within or outside the same cluster) then common TMS nodes are created and the two TMS networks may be linked. For example node in two rulesets may correspond to a question about the state of the same unit e.g. "is the lamp on power supply psu-D red".

In future the TMS may be more closely integrated with the overall control of the blackboard. This means that a TMS network will be set up to maintain the links (justifications) between nodes on the blackboard (i.e. links between fault reports, implicated objects, clusters and hypotheses).

Once again as in the 18 month prototype [17], rules in the various knowledge sources have been written so as to be applicable to types of units. Particular details of instances of units are recovered from a separate frame or object based information source (such as AVALON). This means that the TMS nodes have to be specialised to take account of the instantiation of a generic ruleset to a specific instance. In figure 1 this is shown by the interface to "Knowledge Representation System".

4. INFERENCING COMPONENTS

In this section the various inferencing techniques used by the diagnostic KBS are described, these are MIKIC, TMS and MIKIC/TMS.

4.1. Basic Inferencing

MIKIC was constructed using many of the ideas described by Khong [8]. MIKIC is an object oriented, forward/backward chaining rule based, inference mechanism. MIKIC integrates a rule interpreter with the Flavors object oriented language system such that rules are invoked by message passing just as procedural methods are. MIKIC rules may be grouped in rule-sets where rule-sets may represent a "purpose", a goal or a subset of the knowledge base. Rule-sets may be considered similarly to methods in the object oriented base language ie as procedural attachments to the system objects [16]. In MIKIC there are two types of methods. The first is the procedural Lisp method, standard in the base language which may be defined within an object corresponding to a rule-set. The second type is the rule method which is a rule-set with its own control information and local variables.

MIKIC was constructed as a very flexible tool with a view to its inclusion as a basic building block in many possible future architectures e.g. structured knowledge base, blackboard, distributed process etc. The simple forward/backward control strategy employed by MIKIC has meant that it may be used easily with overall control passed to other systems. MIKIC's integration with other systems may follow an incremental style with systems at various levels tested independently. Further the inference engine may easily be integrated with object representation schemes either directly (using the object oriented base language message passing interface) or indirectly, e.g. via a blackboard.

Rules in MIKIC may be written generally so as to allow them to be applied to data held in separate frame or object based systems, generic rules are also supported (J Biermann (KAE) private communication). The rules may embed calls for information from object representation schemes and external systems as and when required. This makes the rules more generally applicable and therefore more powerful.

Rules may be structured in rule-sets which correspond to individual Flavors [16]. In the prototype KBS described in [17] the structure was mapped onto the structural decomposition (has-parts) of the systems being diagnosed and the types of faults possible. This leads to a methodology for the construction of KBS in large domains where the KBS may be modularised into small and manageable parts. These parts may also be introduced into the active diagnosis threads dynamically only as and when required. This is done by creating an instance of the ruleset Flavor as it is required. This structure will be reflected in the levels and areas of knowledge sources at the hypothesis layer in the blackboard system to be produced during phase 4 of the KRITIC project.

Incremental methodologies for the implementation and test of KBS have been developed [17] which allow systems to be built up as simple (unstructured) rule based systems, then structured rule based systems and then to allow such rule based systems to be integrated into more complex control structures such as blackboard etc.

It was realised through the work in the application domains that while the rule bases and structural and functional information held in the knowledge representation scheme, comprise a considerable amount of pertinent knowledge there were aspects of knowledge and inference capability which were not captured. In particular, there was a need to introduce more flexible control of rule execution and to allow the system to recover properly from inconsistency. Progress in these issues has been made with the introduction of TMS and MIKIC/TMS.

4.2. More Advanced Inferencing

In order to overcome some of the limitations of MIKIC, development of a TMS and its integration with MIKIC to form a hybrid inferencing system MIKIC/TMS were undertaken.

A logic based TMS of the type described by MacAllester [10] stemming from the work of Doyle [5] has been implemented and evaluated. The TMS itself and the ways in which rules are encoded as TMS networks are described in detail by Bigham et al [2]. The TMS has been integrated with the MIKIC inference engine in such a way that backward chaining rules may be more flexibly controlled. In particular non-monotonic actions within the backward chaining rule-set(s) may be handled (by dependency directed backtracking). Of course the TMS may also be used in stand alone mode see 4.2.2.

4.2.1. MIKIC/TMS

MIKIC/TMS is a system where TMS is used principally to provide propositional deduction as a service to the MIKIC inference engine.

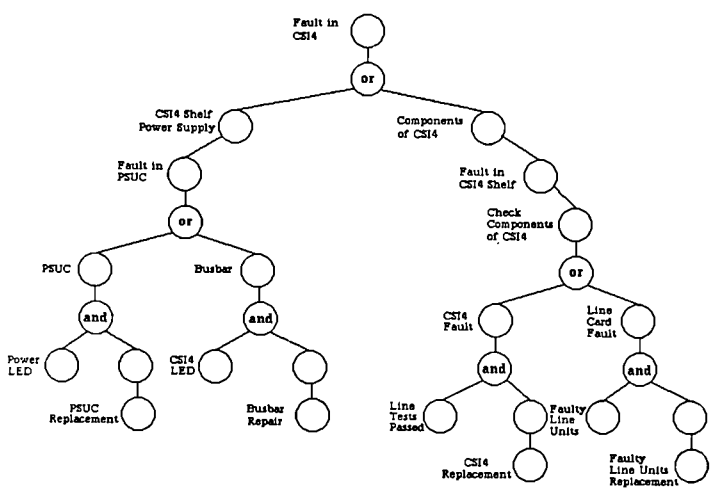


FIGURE 4 TMS Proposition Network for CSI Shelf Ruleset

In order to clarify this use of TMS, the behaviour of the TMS is outlined. The TMS has two major data structures the node and clause. Propositions are represented as nodes, relations between propositions (or nodes) are represented by clauses. For example suppose the propositions A and B are represented by the nodes N1 and N2 respectively, and also suppose that A and B are related by A -> B. The proposition A -> B is represented by another node N3 which may itself have a truth value. The following clauses derivable from consideration of the truth table for -> (implies) relate the three nodes in a network: ¬N3 v¬N1 v N2; N3 v N1; N3 v¬N2. When the truth value of a node is asserted the clauses are used to propagate beliefs through the network.

In MIKIC/TMS when a backward chaining MIKIC ruleset is instantiated (or a number of hypothesis KS in the blackboard system), a TMS proposition network is created. Rules in MIKIC are represented as a clauses in disjunctive normal form. The TMS uses unit clause resolution to perform deduction, whilst incomplete it is quite fast. The nodes in the TMS network may have truth values of true, false or unknown. As a MIKIC ruleset is instantiated, MIKIC/TMS creates the TMS nodes and clauses automatically.

See figure 4 for an example taken from a backward chaining rule set for Monarch maintenance. As the MIKIC backward chaining control fires the rules the results of the actions are used to update the current state of the TMS proposition network. In operation any new relevant propositions may be entered at any time. The TMS will propagate the changes through the network, and MIKIC will modify its behaviour accordingly by skipping branches for which truth values exist. What this means is that new relevant information may be entered into a diagnostic sequence at any stage and may be used to redirect the focus of attention appropriately. Since the control of rule execution is now much more flexibly organised the use of the TMS allows the user to use the KBS in a much more flexible, more dynamic and therefore more user friendly manner. The TMS will of course also allow retraction of truth values to allow non-monotonic reasoning.

Furthermore any contradictions exposed will be identified by the TMS. Currently contradictions exposed have to be resolved by the user by retracting invalid premises via a menu. Future work may allow these contradictions to be resolved automatically. When contradictions arise 'characteristic error' knowledge held in the system will be brought in by TMS and used to resolve the contradictions. The analysis of characteristic errors will use knowledge about the underlying assumptions associated with given premises. When a contradiction is exposed then the underlying assumptions associated with premises may not be valid. An example from the diagnostic domain follows. A test of a hypothesis may involve use of a replacement printed circuit board (PCB). Of course when this PCB is used an underlying assumption may be that the new PCB is working. If the replacement PCB is itself not working then at some point a contradiction will be exposed by the TMS. The underlying assumptions may then be examined and their proper treatment used to resolve the contradiction (and rectify the fault).

Care has to be taken to ensure that the TMS is applied only to a proposition network of appropriate size to ensure satisfactory response times. This is because the computation is potentially expensive. The unit clause resolution is linear with numbers of nodes but processing to handle contradictions by identifying and retracting assumptions is potentially more expensive. However fully acceptable response times have been achieved by focussing attention on a limited number of hypotheses corresponding to possible faults. Furthermore the TMS may be applied to a number of disjoint fault clusters each limited in size which may be executed quasi-concurrently. For this domain it should always be possible to limit the size of the active TMS

network to that which is computationally practicable. Further analysis of the details of the complexity of this program is required.

The TMS has been interfaced to a graphical display which may be used by a system developer to monitor the state of the nodes in the TMS network. This has been used in debugging the TMS and to ensure that the systems developed using TMS and MIKIC/TMS are working properly.

The TMS has been applied to the diagnosis of faults in the Monarch telecommunications switching system (see the example in figure 4). It may be applied to sets of backward chaining rules derived for use (and tested) in the relatively simple depth first strategy employed by the basic MIKIC. This enhances the power of the inferencing system considerably by allowing numbers of diagnostic steps to be shortcut. This MIKIC/TMS integration may be improved if the TMS can be used to direct more flexible forward rule and goal ordering in MIKIC. Work is under way to identify mechanisms by which the costs of remaining goals and sub-goals may be used to optimally redirect the order of goal execution (N.B. this has to be linked to the execution of knowledge sources via the blackboard control).

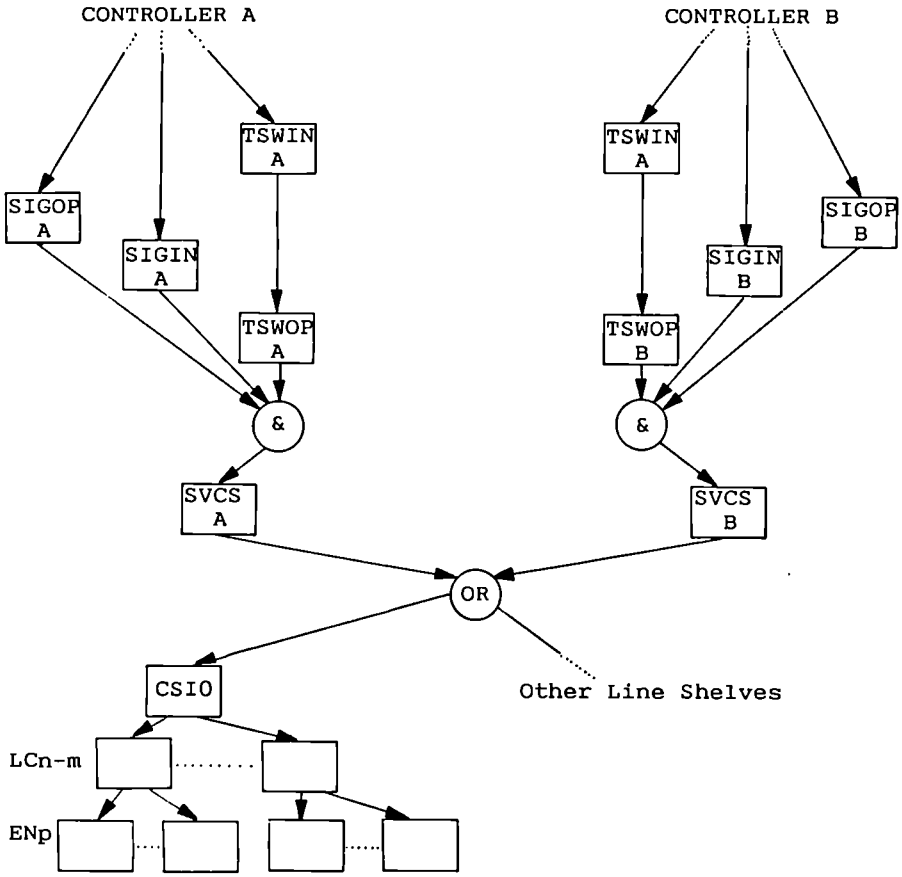
It is obvious that further work has to be carried out but the initial results have shown that the addition of TMS to MIKIC has made the latter much more powerful.

In particular the ability to assert and retract facts will,

1) limit MIKIC's search at run time by eliminating the need to query certain branches,
2) allow the user to input data at any time during a diagnosis thus making data input more flexible and user friendly,
3) may allow the user to dynamically shift the focus of attention,
4) may be used to identify contradictions in rules or in the current state,
5) may be used to permit aspects of non-monotonic reasoning.

4.2.2. The Use of TMS in Functional Dependency Network Modelling

TMS may also be used independently of MIKIC. TMS has been used to handle state change propagation in dependency networks which model an exchange's control behaviour and behaviour under certain fault conditions. Dependency networks associate resources at a particular level of detail by directed linkages. These directional linkages imply dependency, examples are "controlled by" and "powered by". An example network for the Monarch exchange is given in figure 5. Where replication for security (redundancy) is present, this may be handled by the inclusion of the appropriate logical construct in the network (for example "OR" may be used to denote a worker/standby arrangement). The dependency network is used as a TMS proposition network. The TMS will propagate state changes. For example if a resource in the network is not working then none of its dependents can work, similarly if a resource is working we may infer that its parent resource is working also. The TMS will also identify inconsistencies in state, useful in identifying cases where the exchange model is out of step with the exchange itself.



Key:

Generally the functional units higher up the page are the more important for the integrity of the system as a whole. The directed lines indicate dependencies. Dependent units require that their parents are working for their own functionality. Dependencies may be modified by appropriate logical operators, eg 'OR' is used to denote a worker / standby arrangement.

CSIO - Concentrating Shelf Interface O.
ENp - Equipment Number p (customers).
LCn-m - Line Card n-m.
SIGIN - Signalling Input.
SIGOP - Signalling Output.
SVCS - Services.
TSWIN - Time Switch Input.
TSWOP - Time Switch Output.

FIGURE 5 Schematic of Part of Monarch Dependency Network

In work described previously (BALDRIC [17]) state changes were propagated only downwards through the network. The TMS allows propagation of states upwards also. This improves the utility of the model considerably since knowledge about which parts of the exchange are working (which naturally propagate upwards against the direction of the dependency links) is often more discriminating than knowledge about parts not working. For example in diagnosis if we know that a unit is working we may infer that its parent units are working and therefore the search space of non-working units is reduced considerably. This type of state change propagates both working and not working states. This is very useful when a limited amount of information is presented to the user concerning the working state of an exchange. The consequences of that limited information are propagated allowing deduction of a better picture of the overall state of the exchange. Initially this part of the model has been used within a knowledge source at the clustering layer in the blackboard system as part of the clustering and hypothesis generation stage. This model forms a working memory of the current state of the exchange system.

5. KNOWLEDGE REPRESENTATION SCHEMES

Frame systems have a role in representing the relatively well-defined structural and functional information about the domain. Representations of functional blocks and physical parts of the system controlled are required at multiple levels of detail and generality. Representations are required which are able to handle the various replicated elements of the systems controlled. A number of discrete "views" of the exchange system controlled are also required, these include:

views of the physical units e.g. shelves, racks, slide-in-units, parts of slide-in-units, connectors, cables etc;

views of functional blocks of the system which map on to physical units at various levels of detail e.g. controllers, CPUs, line units, and their test and normal functional behaviour (representations of inter-module messages and protocols);

and, views which allow models of the behaviour of the system under fault conditions or maintenance state e.g. the dependency network implemented using BALDRIC [17]. N.B. This dependency network will be generalised to include a greater range of dependencies e.g. powering, control, clock. TMS has been used to implement the state change propagation.

These views have been combined in a single model using a generalised object based representation system called AVALON [20], part of the representation is shown below in figure 6. Such models are used at several stages in the diagnosis KBS. Examples are in symptom clustering and in provision of information required by knowledge sources to test hypotheses. Using this approach the diagnostic rules may be written to be general to types of units with exchange or unit specific data taken from the instances in the knowledge representation scheme. Maintenance of another exchange instance would require only new instances in the knowledge representation scheme. The model may also be used in simulation to provide a training set for use by the rule induction algorithms.

AVALON is a general purpose object oriented knowledge representation system constructed by S. Varey (QMC). AVALON provides the means to construct knowledge representations of objects with relationships such as Sub-class and Instance-of with associated inheritance mechanisms. In addition, more general relationships between objects may be specified by the user to meet

specific user requirements e.g. has-parts, depends-on, powered-by, is-connected-to. Relationships may themselves be organised in hierarchies.

AVALON has also been used to represent some of the behavioural aspects of the PABX such as the message protocols for built-in diagnostic tests.

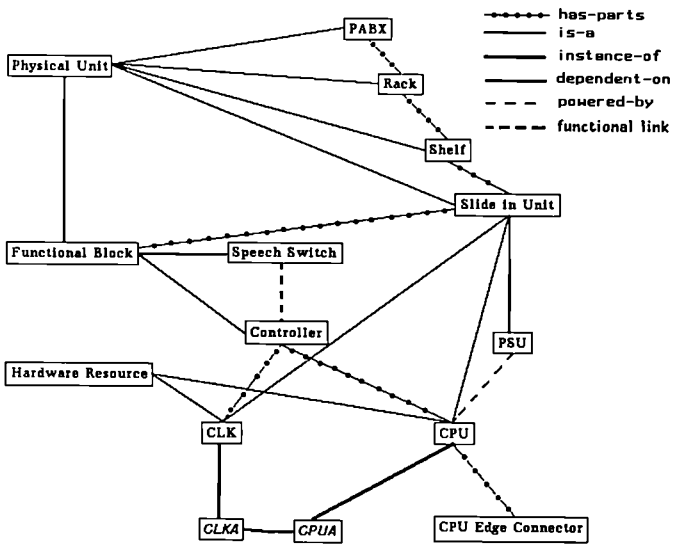


FIGURE 6 Part of Monarch Representation

The model implemented for the Monarch system has used representations of the physical hardware as well as the functional blocks and resources of the system. For example see figure 6 above which shows a small part of the representation scheme. In total some 5000 objects (400 class objects and 250 class relations) and more than 4500 instances (relations and objects) have been represented to date.

The range of object inter-relationships that AVALON supports is powerful. The flexibility with which one can walk through networks using a built-in browser has proved important when trying to construct complex data structures. Some fairly large representation schemes have already been constructed using AVALON, data is input largely via command files. Once stable the representation scheme may be saved and retrieved from dump files which may be either binary or ascii.

Some of the basic node and link display and browsing functions provided by G-Mod (a graphical input and output tool [17]) have been integrated with AVALON. This allows the user to view the object structures he is creating in a much more direct manner than is possible with the AVALON browser. G-Mod diagrams may be produced which are similar to those with which the user is familiar. This greatly eases data input and consistency checking. The representation scheme may also be browsed using V-graph [19] which allows networks to be displayed as a tree displayed on a virtual screen.

6. OFFLINE RULE LEARNING

Knowledge acquisition in domains such as telecommunications switching operation and maintenance is a difficult problem. Some of the reasons for this difficulty are given in [17]. In order to alleviate some of these problems techniques have been developed by which "design" knowledge can be used to induce rules and thereby automate some of the knowledge acquisition.

Simulations of telecommunications switches may be used to generate examples linking fault symptoms to faults. These examples may be used as input to "learning" programs. N.B. the purpose of applying the induction algorithms to the training sets created through the simulations is to summarise in relatively simple rule form the results of many simulations.

Other workers [3], [15] have suggested the need to use models of system behaviour as the basis for KBS, such systems are often called second generation expert systems. In the telecommunications switch case data about the design is more readily available than heuristic knowledge, particularly when systems are deployed before experience is gained in their use.

Two approaches have been studied, one using simulations of the built-in diagnostic tests in an exchange and the second using functional dependency networks. The induction algorithms used are similar to ID3 [13]. Two "evaluation function" algorithms used have been based on chi squared and Gini[4]. This Gini algorithm also incorporates probabilities based on the a priori probability of failure of unit(s) (related to the mean time between failure of the unit(s)). A bottom up approach to the learning work has been taken where the results of a large number of experimental runs on various data appropriate to the domain are taken and analysed. The details of this work and the algorithms used will be described elsewhere, but some of the domain specific aspects are discussed below.

6.1. Rule Induction from Simulations of Built-in-tests

In the first approach the design information includes data concerning the objects (including communication links) which make up the functional blocks of an exchange, and the messages and their protocols which make up the built-in diagnostic tests of the system. The success or failure of these messages has been made conditional on the working state of the blocks. Such a model may be run in simulation mode to produce examples of symptoms (tests failed and passed) for simulated faults (both singly and in combination). The examples may then be used as input data for an induction algorithm.

G-Mod [17] has been used to capture design knowledge about units and their interconnections, diagnostic tests and normal function task protocols. This information is now encoded in AVALON, it may be used to model behaviour of the system, in particular the relation between fault symptoms and suspect units. Fault symptoms are typically the result of a failed background test (part of the built in test equipment of the exchange) or a call failure. These may be simulated using G-Mod as sequences of messages passing through the various functional objects of the system.

The induced rules link the symptoms (in the form of observable diagnostic test results) to suspect parts of the exchange (PCBs, connectors, cables etc).

This work is still at an early stage of development, however a number of possible applications are foreseen. This algorithmic approach to knowledge acquisition for very large problems may be more applicable than the more conventional KBS methods. The possible applications include the following:

finding the optimal order for the application of diagnostic tests - that is applying the next most discriminating test, given the result of previous tests;

looking at the correlation between tests to identify tests which are redundant and parts of a system not fully covered;

looking at the correlation between faults - so that when the most likely fault is found not to be applicable the next most likely fault can be identified and considered.

This approach requires a large amount of data about the structure and functional information of an exchange. A knowledge base of this kind requires effort to construct. To be fully cost effective the induction system will have to be linked to the CAD system used to construct the original system. Nonetheless the results described here have demonstrated the viability of the approach.

6.2. Rule Induction from Functional Dependency Network Models

The second method uses the more abstract notion of dependency network as the basis for simulation (see figure 5). Similar approaches to this have been described by other workers [9]. Such dependency network data is more compact than a full functional model as above and may be readily available from exchange data builds [1]. One or more faults may be simulated in the model and the symptoms generated are defined by the states of the observable resources (only certain of the resources are denoted observable). The rules identify the possible areas where faults may be present considering both single faults and double faults, in future this will be extended to handle multiple faults generally. One problem associated with this work has been the need to identify the states of all observable resources as input to the induced rules, in addition it is not desirable to input impossible states to the induced rules, what may be done is as follows. Certain of the known resource states may be input directly to the TMS model. The TMS will deduce other states as a result of this. The TMS will also prevent inconsistent and impossible states. The unknown states remaining are assumed working (since no fault reports have been received from them).

The simulation and rule generation is important to ease the acquisition of diagnostic rules from more readily available design information. Rules linking symptoms to simulated faults were generated from a simulation containing the exchange's resources. The rules form the basis for some of the clustering knowledge sources in a complete diagnosis system. This use of design information is important where no single expert is likely to have all the necessary knowledge and exchanges have to be maintained from the day they are deployed, before experience is gained in their use.

7. DISCUSSION OF RESULTS - LESSONS LEARNT AND FUTURE WORK

The blackboard architecture allows flexibility in control. It allows the integration of multiple knowledge sources which may co-operatively solve a specific problem. At the implementation level it allows development of systems in a modular, incremental and iterative fashion. These techniques have proved appropriate to the telecommunications switching domain diagnosis problem. Future improvements and generalisations of the blackboard system will be sought. This will include: examination of the organisation of KSs, exploitation of parallelism, examination of alternative control strategies, and provision of help to the systems user. Such future work is linked to the work of a control task force within the consortium where blackboards are being compared and contrasted to another approach called CELL/TISSUE.

The TMS is a general tool with a good theoretical foundation. It is a good prototyping tool, with an underlying network representation which is easy to understand. The TMS is applicable in a number of areas in different ways e.g. MIKIC/TMS, and in the propagation of fault states in a functional dependency network. Future work in this area includes the incorporation of: the capability to handle numerical values, error analysis when contradictions are exposed, methods of representing persistence (over time) of truth values. In future it is also possible that an ATMS approach will also be considered.

MIKIC/TMS was developed to overcome some of the limitations apparent in the early MIKIC work. In particular it allows flexible handling of input of information to MIKIC rulesets. It also allows proper recovery if something inconsistent happens or if beliefs are changed. Future work will allow more general translation of MIKIC rulesets to a corresponding TMS network representation. It is also possible that more dynamic control of MIKIC goals and sub-goals will be attempted to take proper account of information currently available.

Although the work on rule learning is still at a relatively early stage it holds considerable promise for the automation of knowledge acquisition. This is particularly valuable for large 'designed' systems where specific heuristics may not be available and no single expert exists. Good progress is being made on induction of rules for clustering fault symptoms. Some of the early work has produced rules which have been incorporated into the blackboard based KBS as clustering KSs. Future work will attempt to define more general rules and methodologies.

AVALON has allowed the construction of a large but economic model of an exchange comprising a number of 'views' of the system at multiple levels of detail and specificity. Such a model forms the basis for existing, mixed heuristic and design-based rule induction approaches to diagnosis. In future the model may form the basis for experiments on other more general approaches to switch diagnosis and for other aspects of the tele- communications switching domain e.g. operation and design. Future work on AVALON includes partitioning and dumping of the database to allow much larger models to be accommodated and improved knowledge representation management functions.

8. CONCLUSIONS

Considerable progress has been made both in the definition of an architecture and KBS development for solution of diagnostic tasks in telecommunications switching systems. Considerable progress has also been made in the development of software tools and techniques to allow such developments. These tools have been developed entirely within the consortium as part of the KRITIC project.

Work will continue on the production of a demonstrator KBS based on the work described above which is expected to be complete in September/October 1987. The KRITIC project finishes in December 1987.

ACKNOWLEDGEMENTS

The authors acknowledge the many valuable discussions held with all members of the consortium which have helped in this work. Special thanks are due to: Vee H. Khong (QMC) who worked on MIKIC and MIKIC/TMS; Erick Gaussens (Framentec S.A.) for comments and suggestions on the early blackboard work; and, Stephen Varey (QMC) for work on AVALON.

REFERENCES

[1] Baty, R.M. and Sandum, K.N., System X: Maintenance Control Subsystem, British Telecommunications Electrical Engineering Journal Vol.3 Pt.4 (1985).
[2] Bigham, J. et al, Inference Tools for KBS in Industrial Application Areas, to be published (1987).
[3] Bratko I., Mozetic I., and Lavrac N., Automatic Synthethis and Compression of Cardiological Knowledge, Expert Systems, Edited by J.M. Richards.
[4] Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J., Classification and Regression Trees (Wadsworth International Group, Belmont, Calif. 1984) pp. 93-129.
[5] Doyle J., A Truth Maintenance System, Artificial Intelligence Vol. 12 No. 3, (North-Holland 1979).
[6] Goyal, S. et al, Compass: An Expert System for Telecommunications Switch Maintenance, Expert Systems Vol. 2 No. 3 (1985) pp. 112-126.
[7] Hayes-Roth, B., A Blackboard Architecture for Control, Artificial Intelligence Vol. 26 No.3, (North-Holland 1985).
[8] Khong, V.H., A Study of Rule Based, Frame Based and Constraint Representation in AI, Ph.D. Thesis (University of London, 1985).
[9] Kramer M.A. and Palowitch B.L., A Rule Based Approach to Fault Diagnosis Using the Signed Directed Graph, AIChE Journal (1986).
[10] McAllester D.A., An Outlook on Truth Maintenance, AI Memo No. 551, (MIT 1980).
[11] Penny Nii H., Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, AI Magazine Summer, (1986).
[12] Penny Nii H., Blackboard Systems: Blackboard Application Systems, Blackboard Systems from a Knowledge Engineering Perspective, AI Magazine August (1986).
[13] Quinlan, J.R., "Learning Efficient Classification Procedures and their Application to Chess End Games.", Machine Learning: An Artificial Intelligence Approach, R.S. Michalski et al. (Tioga,Palo Alto, 1983).
[14] Seviora, R.E., Field Debugging of Telecommunication Switching Machines, Globecom 83 (IEEE Global Telecommunications Conference 1983) Vol. 3 pp. 1480-1485.
[15] Steels, L., Second Generation Expert Systems, Future Generations Computer Systems Vol.1 No. 4, (North-Holland 1985), pp. 213-221.
[16] Weinreb, D. & Moon D., Lisp Machine Manual, (MIT, Cambridge, Mass 1981).
[17] Williamson, G.I. et al, Using a KBS in Telecommunications, Proc. ESPRIT Conference 1986 (North-Holland 1986).
[18] Wittig, T., Power Systems fall under KRITIC's Eye, Modern Power Systems, (London 1987).
[19] Wittig, T., Virtual Graphic Browser, Esprit Project 387, Standard Deliverable 5, pp. 47-54.
[20] Varey, S.R., Ph.D. Thesis (Univ. of London, in preparation).

Project No. 280

KNOWLEDGE REPRESENTATION FOR INTELLIGENT HELP SYSTEMS

John van der Baaren

Courseware Europe BV, Ebbehout 1, Zaandam, The Netherlands

The provision of intelligent help to a user of an interactive information processing system (IPS) requires the ability to follow and interpret the user's interaction with the system and actively support the user's task performance. In this paper a framework is presented for representing the knowledge about an IPS. This framework supports the planning, plan recognition and explanation giving in an intelligent help system.

1. INTRODUCTION

The overall goal of the EUROHELP project is to provide a development environment for intelligent help systems for interactive information processing system (in the remainder of the paper abbreviated as IPS), like editors, spreadsheets etc.. The function of a help system is to provide information about the IPS when the user needs it. This need can be expressed by the user, but also inferred by the help system itself (Breuker (1)). The representation of the knowledge about the IPS is the basis for a help system for a specific IPS.

The representation of the knowledge about the IPS must support very divers functionality present in a help system. Different types of questions must be interpreted and suitable answers constructed. E.g. "How do I delete the last word on this line?", "What is a binary file?", "What is the difference between print and list?" etc.. The user's interaction with the IPS must be interpreted. The current state of the system must be accessible and explainable. Inefficient or erroneus user actions must be discovered and interpreted in terms of lacking knowledge or misconceptions. It must be possible to assess and update a model of what the particular user knows about the IPS. Moreover it must be possible to construct from the representation a curriculum for learning the IPS and to find suitable occasions for teaching.

Furthermore there are a number of "non-functional" requirements for a proper representation of an IPS. It is important to prevent that similar knowledge is represented twice, e.g. a component that constructs the answers to "How do I ....?" questions should use the same representation of plans as a plan recognition component. A requirement for the final help system development environment would be that it is possible to construct a representation for a specific IPS without a profound understanding of the principles underlying the intelligent help system, being a very experienced user of the IPS should suffice.

During the pilot phase of the EUROHELP project a prototype of an intelligent help system was developed for a specific IPS (Unix mail). While satisfying most of the functional requirements mentioned above, the non-functional

* The research reported in this paper is funded by ESPRIT as project P280. Partners in the project are CRI A/S (Denmark), DDC (Denmark), ICL (UK), University of Leeds (UK), University of Amsterdam and Courseware Europe (The Netherlands).

requirements were more problematic. The research since aims at providing the three essential components for constructing a knowledge representation of an IPS. The first component is the knowledge representation environment, which can be seen as a specialized knowledge representation language that already contains the representations of the concepts and relations that form the generic basis of information processing systems. Much research has been conducted on this subject within the EUROHELP project over the last two years (Duursma (2), Holgaard (3), van der Baaren (4)). Secondly tools are needed that support the actual construction of a representation for a specific application. Various tools are being considered at the moment, ranging from editors, graphical browser, libraries with concept definitions found in many domain (e.g. menus, command line, mouse), consistency checkers, compilers etc.. The last component is the methodology that guides this specification process. This paper concentrates on the first topic and only touches upon the other two.

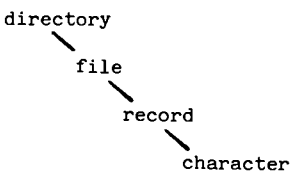
2. AN EPISTEMOLOGICAL DESCRIPTION OF INFORMATION PROCESSING SYSTEMS

At a first glance making a representation of an IPS doesn't seem to be much of a problem. It is always possible to give a complete description of an IPS; it's a closed world. One can even say that a perfect representation already exists in the form of the program itself and the interpreter. There are however a number of problems with this view. An IPS is a closed world, but a representation of the knowledge about it is not. An unlimited number of different models are possible. Moreover a complete representation of what you can do with an IPS is not possible, e.g. an IPS might be used for tasks the designers had not in mind. Although the sourcecode of an IPS is a perfect representation, it's computer language dependent (at least) and therefore not suitable as the basis for the construction of help systems in general. Moreover when using the source code there will be a considerable distance between the user's intentions and the representation of the IPS.

In this chapter an overview is given of the abstract, generic concepts that we think are needed to describe the functionality of any IPS (Breuker & de Greef (1)). In chapter 3 we consider another point of view and try to represent in a general way how an IPS is used when performing a task.

2.1. Objects

The most important class of objects in any IPS are information containers. Information containers are organized in a consist-of hierarchy and are usually ordered. An example:



In the end each information container contains an ordered set of primitive objects (generally characters). Each program, sub-program or mode can have it's own (organisation of) information containers. Information containers have additional information associated with them, e.g. the size, the name and the creation date of a file. When this information can be used to reference a specific information container we call it an identifier, e.g. the name of a file.

The set of information containers constitutes only a small subset of all the objects that are manipulated in an IPS. Dependent on the specific application a considerable amount of objects which are dynamically constructed when an action has to be carried out (word, sentence, cursor_to_end_of_word etc.). We want to be able to talk about these objects too; we want to be able to describe, or at least name, the objects that are manipulated. We refer to these dynamically constructed objects with object references.

Classes of object references are organized into an hierarchical structure, organized according to increasing constraints upon the class of objects described. In Fig. 1 and example is given of a part of such a structure.

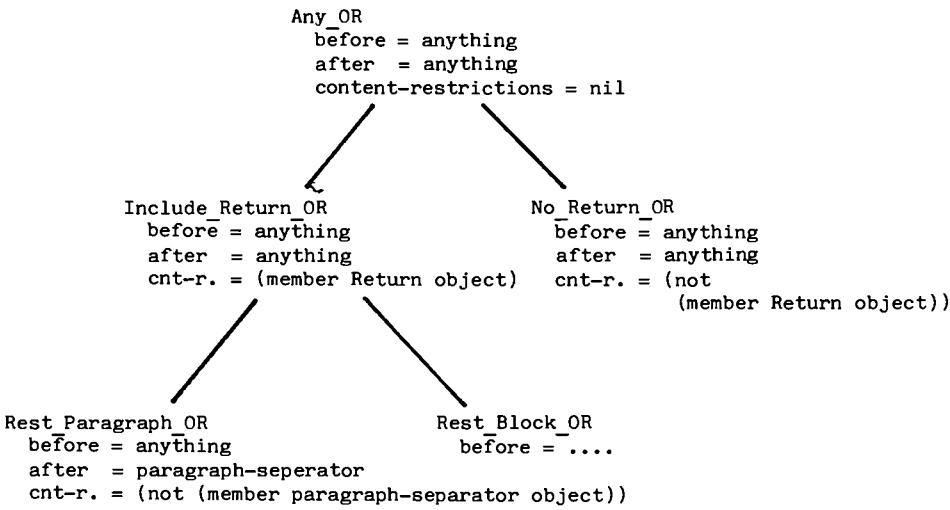


Fig.1 An example of object references. OR stands for object reference. The definition is given in terms of constraints on the objects before and on the objects after the object reference, and a set of restrictions on the primitive objects contained in the object.

This structure would be sufficient when system procedures only operated on one object at the time. To include for example global replace commands it is necessary to have references to sets of objects. Specialisations can be added for other purposes as well. For instance we might want the user to be able to refer to these "objects" when interacting with the help system or we might want to register a specific limited use of a system procedure. E.g the represention for the specialisation "last word on a line":

```
Last_Word_OR
  before = word_sep
  after = line_sep
  content-restrictions = (not (member wordsep object))
  explain = "the last word on a line"
```

Now we can handle questions like "How do I delete the last word on a line?" even though there is no command that manipulates this specific kinds of objects. By moving up the constraint hierarchy we can find object references that have a system procedure associated with them. This way we are able to

give answers like: "You can use the D-command to delete the rest of the line." or "To delete a word move the cursor to the beginning of the word and type "dw.".

2.2. Actions

In the previous section we dealt with the representation of the objects in an IPS, in this section the other major class of concepts is described: the actions that are performed on the objects by an IPS. To see where actions fit in with our representation I'll give some informal definitions of command and system procedure. A command is the activation of a system procedure, while a system procedure is the representation of the effect a command can have. Effects are described as sequences of actions on objects.

It is at the level of actions that we try to provide the generic building blocks for representing the effects of system procedures in general. The key idea is to provide a relatively small set of primitive actions that are sufficient to describe the effect of any system procedure. For example to select, create, delete, show or change an object. Each primitive action is defined in terms of a before and after state:

```
Primitive_action
  IsA: Action
  Name: (a string)
  Before_state: (a system_state)
  After_state:  (a system_state)
```

The primitive actions can be used to build more complex or specialized actions. These are always defined in terms of the primitive actions:

```
Composed_action
  IsA: Action
  Name: (a string)
  Effect: (a primitive_action_sequence)
```

For example a "move" could be defined as a composed action which has as effect the primitive actions "delete" and "create". Actions can be refined by placing restrictions on the type of objects they take as arguments. E.g. "send" may be a specialisation of "copy" in the case of messages. The primitive actions enable use to include general inference capabilities in our help system development "shell" (e.g. for planning or emulation). The composed and specialized actions are more suitable to provide concise and well explainable effect descriptions of commands.

3. GENERAL STRUCTURE FROM AN OPERATIONAL PERSPECTIVE

Undoubtedly the most important perspective on the representation of the IPS is that of task performance. An intelligent help system must be able to do both planning (e.g. for answering enablement questions) and plan recognition (e.g. for recognizing inefficient task performance) with the IPS. From this operational perspective the user is performing tasks by means of plans. So the basic items in our representation will be tasks and plans. The user's task performance consists of entering a context, this can be a program, subprogram or mode. In this context one or more tasks are performed that are specific for this context. Each such task consists of the application of one or more system procedures. Applying a system procedure entails the application of one or more interaction tasks (pressing a key, clicking a mouse etc.).

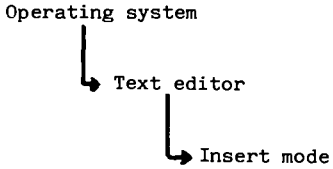
So the user's task performance can be described at four levels of abstraction:

A. mode_Context_Task (e.g. Change text using editor)
B. mode_Task (e.g. Delete a piece of text)
C. system_Procedure_Task (e.g. Apply delete_word system procedure)
D. interaction_Task (e.g. Type "dw")

Tasks at the levels A to C can be refined to plans consisting of subtasks. For each task all possible refinements are represented. In addition it is specified which refinements are applicable in a specific situation. This structure has the following characteristics. First it's hierarchical in the sense that going up the levels provide a more abstract description of the same task. Second the description at each level is complete. Third all subtasks can be further refined independently. In this way it is not necessary to refine a task all the way down to the interaction level. It also provides the possibility of mixing the descriptions of the different levels. For example: "Move the cursor to the line and type "dd"", which mixes a mode_Task description of the first subtask with an interaction_Task description of the second.

3.1. Mode context tasks

A mode context task is the top level description of the user's intention. This follows the top level breakdown of functionality as provided by the IPS in terms of programs, sub-programs and modes. Contexts can be organized hierarchicaly and entering the succesive contexts is like zooming in on the information containers. E.g.:



With each mode_Context_Task a set of all possible mode_Context_Plans is stored, together with a ruleset that selects the applicable plan(s) in a specific situation. A plan specifies an ordered sequence of subtasks (mode_Tasks in this case) to be carried out.

3.2. Mode tasks

At this level the basic functionality of the context (i.e. program, sub program or mode) from the point of view of the user is represented. It can be seen as a description of the operations the user wants to have carried out on one or more objects. Each task is an abstraction from a set of system procedures. From a planning point of view we need this level to represent how the user can use the IPS in task performance. From a plan recognition point of view we need to be able to track down the users intentions. An example of a mode task:

```
Delete_Object
  IsA: mode_Task
  Operation: delete
  Arguments: (an object_reference)
  Planset: "A list of mode_Task_Plans"
  PlanRuleSet: "A ruleset to pick a plan"
```

Operations are abstractions of the effects of system procedures, comparable to the primitive and composed actions described in 2.2.

The representation at this level is restricted to the set of tasks which is needed to cover the basic functionality provided by the system procedures of the IPS. When every system procedure has a task associated with it at the mode task level we consider the level to be complete. Further extensions are always subjective. It is up to the help system developer when making a help system for a specific application (and maybe a specific user population) to determine how many, more complex, tasks to include.

3.3. System procedure tasks

The set of commands available with an IPS determines completely the possibilities of the system. However they are not suitable as a level of abstraction as such. A command can have different effects in different situations (e.g. the delete-character-command in the VI editor normally deletes a characters and moves the cursor to the next, but at the end of a line the cursor is placed at the previous character). In some cases different commands have exactly the same effect (e.g. the type and print commands in Unix mail both display the contents of a mail message on the screen).

Instead of using the commands we choose to decompose the mode tasks in terms of system procedures. A system procedure is the effect of a command in a well defined set of situations. A system procedure task can be defined as:

```
Name_Of_Task
  IsA: system_Procedure_Task
  Effect: (a system_procedure)
  Arguments: (an object_reference)
  Plan_set: "a list of system_Procedure_Plans"
  Plan_Rule_set: "a rule set to choose a plan"

Name_Of_Plan
  IsA: system_Procedure_Plan
  Command: "a command"
  Arguments: "the arguments for the command"
```

Attached to the plan is also the procedural knowledge needed to determine the value of the arguments for the command from the actual object references given in the task. Also the inverse operation is supplied in order to enable plan recognition. E.g. a search command in a text editor takes as arguments a string and the options ignore_Case and only_Whole_Words. The effect of the system procedure we aim at might be described as: place the cursor at the first character of the object reference and update the screen. In this case the relation between the object reference in the system procedure task and the arguments supplied to the search command is quite complicated.

The structure of the actual interaction between user and IPS is represented with the commands. Combining this information with the system procedure plan yields a sequence of interaction tasks. E.g.:

```
(Display: Prompt)
(Type: "delete")
(Type: Space)
(Type: "file_name")
(Type: Return)
```

3.4. Interaction tasks

At the interaction level a description is given in terms of a sequence of the physical user and system actions. The structure and content of the dialogue between user and system is defined by the commands. The basis for our description of the interaction between user and system was Moran's (6) Command Language Grammar.

At the level of the system procedure tasks it is only specified how the command will be interpreted, i.e. what will be the effect of executing the command. What still needs to be represented are the order in which the various parts of the command should be specified, the physical user actions that are required to specify the parts, the systems prompts and responses to the different parts of the command. Moran (6) describes a set of generative rules that needed only minor adaptations to suit our needs.

The plan of interactions tasks covers all actions of both user and system until the interpretation of the command. Next the system procedure is executed. This is the whole effect of the command, including the necessary interface actions, i.e. redrawing the screen, beeping etc..

The representation outlined above can be used for both planning and plan recognition. Moreover the levels of abstraction give a large flexibility to the explanation facilities.

3.5. Alternative approaches

The representational framework outlined above has some disadvantages. For instance all possible plans for a task are explicitly enumerated including the knowledge needed to be able to select the correct plan in a given situation. For a project which develops a general help system development system a more generative approach to planning and plan recognition seems to be preferrable. To justify our approach an alternative is described and the kinds of problems involved shown.

First the representation of an IPS is separated in two parts: the technology space, which is a pure and objective description of the IPS and the task space, which is represents what the system can be used for in terms of real world task. The task space can be seen as a large and/or tree where the leaves of the tree have a description of the task in terms of before and after state. Given this description a general purpose planner could browse through this objective system description and come up with a plan. A problem with this is that most system procedures will have effects that are not described in the task. E.g. a command that deletes a line of text might also put this text in a undo buffer, update the screen etc.. It turned out to be impossible to give general guidelines about which effects can be ignored and which effects should be taken into account in planning. The only way out is to intertwine the subjective task space and the objective technology space. For instance by including relevant effects of system procedures in the state descriptions of the tasks or by typing the effects of system procedures as intended or side effects as Brachman (7) does with his representation of the Hermes mail system. By including the mode_Task level in our system description we claim to have found a cleaner solution, that is also much easier to apply when constructing a representation of a specific IPS.

4. CONCLUSION

A representational framework was presented that is generic for information processing systems and is powerful enough to support the functionality required of an intelligent help system. Planning and plan recognition are supported using basicly the same datastructures. It also seems feasible to associate additional knowledge with the generic concepts in the representation. This enables diagnostic modeling of the user, to find a lack of knowledge or misconception (van der Baaren (8)). Also it is possible to generate answers for different types of questions (Hartley (9)) and to construct automatically a curriculum for teaching the IPS (van der Baaren (4)).

Building a representation of a specific IPS along the lines described in this paper is very complicated and probably not possible without a thorough understanding of the functioning of an intelligent help system and the general knowledge representation issues involved. Because the Eurohelp project aims at developing a help system development environment that can be used without specialized knowledge in these areas we now concentrate on tools and a methodology that support the knowledge representation process.

REFERENCES

(1) Breuker, J.A., A Shell for Intelligent Help Systems, IJCAI 87, 1987.
(2) Duursma, C. & Maas, S., A report on the Development of a domain representation for EUROHELP, University of Amsterdam, 1986.
(3) Holgaard, L., Mortensen, P. & Stausholm, L., Domain Representation, Report CRI/EUROHELP/046, 1986.
(4) van der Baaren, J, Duursma, C., Hijne, H., Romeijn, T. & van der Vlugt, D., Generality Test, Report COE/EUROHELP/021, 1987.
(5) Breuker, J.A. & de Greef, P.H., Information Processing Systems and Teaching & Coaching in HELP Systems, deliverable 12.1, Esprit project 280, 1985.
(6) Moran, T.P., The Command Language Grammar: a representation for the user interface of interactive computer systems, International Journal of Man-Machine Studies, 1981, 15, pp. 3-50.
(7) Brachman, R.J., A Structural Paradigm for Representing Knowledge, BBN report 3605, 1979.
(8) van der Baaren, J., Diagnostic Modeling in Intelligent Help Systems, Report COE/EUROHELP/020, 1986.
(9) Hartley, J.R., Smith, M.J. & Carr, I.G., The Rationale of Explainer, Report ULE/EUROHELP/024, University of Leeds, 1986.

Project No. 280

Coaching Strategies For Help Systems: EUROHELP

Joost Breuker, Radboud Winkels, Jacobijn Sandberg

University of Amsterdam, Dep. of Social Science Informatics
Herengracht 196
1016 BS Amsterdam, The Netherlands
breuker@swivax.uucp

Abstract

The research reported here is part of a project aimed at the construction of an environment for building intelligent help systems. Core of this environment is a shell that contains all domain independent knowledge and procedures. A help system supports the user in handling and mastering an information processing system. Therefore, it should not only answer questions of users, but it should also 'look over their shoulders' and interrupt when appropriate. Part of the shell and focus of this paper is a generic coach. In a help system a coach has two functions: to assist the user with a current problem and to teach the user about the information processing system.

1. The EUROHELP Project.

The research reported here is part of the EUROHELP Project *) This project is aimed at the construction of an environment for building intelligent help systems for 'information processing system' (IPS). Core of this environment is a shell which contains all domain independent procedures and knowledge. The major task of a developer of a help system for some specific IPS will be to load the shell with a representation of the domain concepts (tasks, objects, commands, syntax, methods of object reference, etc). It is expected that the development of an IPS will proceed in parallel with the development of its help functions, but providing help systems as 'add-on's for existing IPS is more of a challenge which we have taken up at the start of the project. It is often believed that well designed IPSs have such a self-evident structure and such 'understandable' (metaphoric) interfaces that help seems superfluous. A good user interface is supposed to have help implicitly wired in. For instance, the famous Apple Macintosh interface looks so self evident, that novice users soon feel quite comfortable. However,

*) This research is partially funded by the ESPRIT programme of the European Community under contract P280. Partners in this project are: CRI (Denmark), DDC (Denmark), ICL (U.K.), University of Leeds (U.K.), Courseware Europe (Netherlands) and University of Amsterdam (Netherlands). It encompasses an effort of about 100 man years over a period of 5 year, of which 2 have been spent now.

experienced users complain about its modularity and inaccessibility of its elementary processes. In other words, the Mac is hiding the fact that the electronic world is differently shaped from the real world. The real world metaphors break down very soon and even impede the acquisition of skill and insight into an IPS. More detailed reasons why help systems will always be needed is presented in Breuker (in press).

The EUROHELP shell is under construction now; specifications have been written. The specifications are based upon experiences with building a prototype add-on help system (EUROHELP.P0) for Unix-Mail (Breuker, in press). Implementation is in LOOPS on a Xerox 1186, which is connected to a system running Unix. The specifications are moreover based upon empirical studies in various IPS domains (e.g. the Unix-Vi editor, MacWrite, spread-sheet packages, etc.), using i.a. mock up studies in which human experts act as coaches who communicate with various types of users via terminals (e.g. Winkels et al., 1986; Breuker et al., 1987). These experts monitor the performance of the user, interrupt for advice, and answer questions of the user. These empirical studies are supported by more analytic studies on knowledge representation of IPS, user modelling, planning, diagnosis, coaching and question interpretation, for which prototype modules have been developed or will be developed to support the construction of the shell. The EUROHELP shell will be constructed in a more transportable implementation formalism (CommonLisp/-Loops). In this article we will focus on the construction of one of the components of the shell: a generic COACH, Before discussing this COACH, a short overview of the shell is presented (see also Breuker et al., 1987).

2. What's in a help system.

The function of a help system is to provide information about the use of some IPS when needed by any type of user. The need can be expressed by the user or inferred by the help system. This means that a help system should have the role of a human coach, who looks over the shoulder of the user to interpret performance, who interrupts when things go wrong or when there is an opportunity to extend the repertoire of the user, and who is able to answer questions in the context of current use of the IPS.

Monitoring on line the performance of the user entails many conceptual and computational problems, but these are not qualitatively different from those in intelligent teaching systems in general. Because user needs may either be identified by the system or by the user, EUROHELP contains a QUESTION INTERPRETER and a PERFORMANCE INTERPRETER, or a passive and an active side (Fischer et al., 1985). Both communicate with a DIAGNOSER which tries to identify the actual ('local') need of the user. This local need can always be stated in terms of a specific lack of knowledge or misconception. This lack of knowledge may not only refer to concepts, but also to current states of the system. Many local needs are identified by the fact that the user makes an error.

The tracing of errors in using an IPS is simplified to some extent by the fact that many erroneous actions of the user are not executable. However, the variety of causes of executable errors is as complicated as in any 'natural' domain, because users may acquire all kinds of misconceptions, ranging from wrong models of the underlying 'virtual machine' to not knowing about a side effect of some command. The identification, or diagnosis, of problems of users is not only complicated by the fact that lack of knowledge may lead to executable errors, but particularly by the fact the system has to infer what the intentions of the user are. The

system can look over the shoulder of the user, but it would become a real nuisance if it would ask what the user is up to all the time. In intelligent teaching systems this is a minor problem, because the system itself prescribes what goals the user should accomplish: it presents specific problems to the user/student, and makes safe assumptions about at least the global goals. This means that in a help system there is a great emphasis on plan recognition in interpreting the performance of the user. Because the *skill* (not the insight) in handling an IPS consists mainly of planning actions, a planner is also an important component of the performance interpreter. The planner may work in close cooperation with the plan recogniser to provide top-down constraints on the interpretation of user intentions, but may also work independently. This is for instance required when the user makes an executable error and an unintended state is created. Help does not only imply to tell the user about what a correct action would have been, but also how to undo this current state. Figure 1 gives a global view of the architecture of a help system.

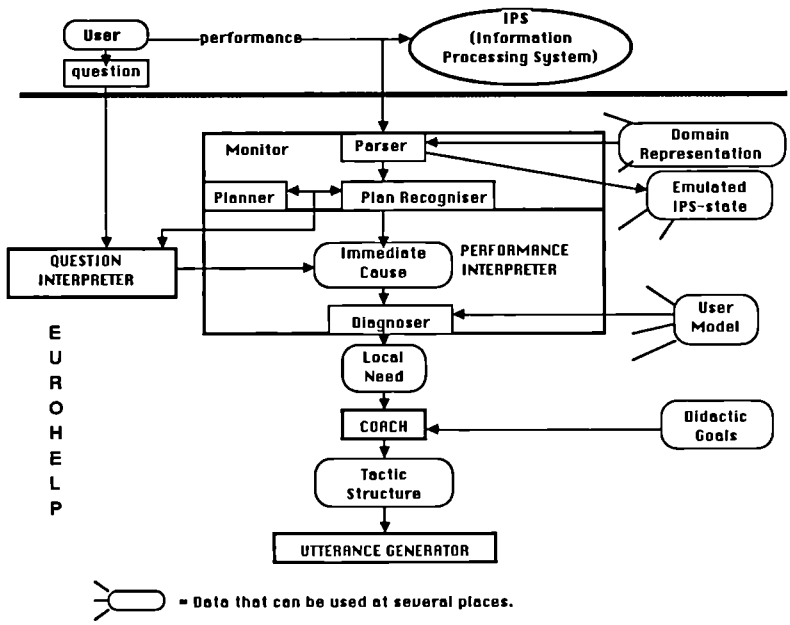


Figure 1 : *Architecture of EUROHELP*

3. The COACH.

Once a problem (local need) of a user has been identified, the COACH comes into action. In a help system a COACH has two functions: to assist the user with a current problem and to teach the user about the IPS. These two functions support one another: correct performance facilitates learning; knowledge about the IPS enables (better) performance. However, the scope of these functions is different. Learning goals are long term goals, while the 'HELP' function is a very local one. Therefore we distinguish 'global needs', i.e. the knowledge to be acquired about a particular IPS, and *local needs*, which state the current problem of the user. Whenever a local need can be related to a global need, i.e. the user is supposed to be able to learn from the information presented, the COACH should teach; otherwise it simply presents the required information (HELP), without expecting this information to be remembered.

Presenting information consists of a sequence of 'speech acts' or *tactics*. This sequence is the result of a planning process that takes into account what to say when and how, given the identified problem of the user (i.e. the local need). This is what a 'teaching strategy' is about. Current intelligent teaching systems (ITS) contain more or less fixed, prewired teaching strategies (e.g. Sleeman & Brown, 1982; Self, in press). In the EUROHELP.P0 -a help system for Unix-Mail- coaching strategies were also prewired in the form of fixed frames in which topics (what to say) could be inserted. However, the large variety of potential local needs requires a more generative approach. As literature (e.g. Ohlsson, in press) and empirical data show: there are no *fixed* coaching strategies. They are flexibly generated as a function of the current problem and state of knowledge of the user. Therefore, there is a very recent tendency to construct flexible, multileveled 'discourse planners' for intelligent coaching systems (e.g. Elsom-Cook, 1987; Macmillan, 1987).

The structure of the COACH consists of the following three layers, which are similar to those proposed by Woolf & McDonald (1984), but with more functional differentiation.

(1) The DIDACTIC GOALS. Didactic goals are concepts in the domain representation, structured in such a way that they provide a didactic view on the domain. They form the curriculum in the same way as genetic graphs (Goldstein, 1982) specify sequence and (didactic) relations between topics for teaching. Because the HELP system has to serve all types of users (from naive to expert) with widely different states of knowledge, the DIDACTIC GOALS are *generated* dynamically on the basis of the current state of the user model and principles of 'optimally effective learning' in which the distinction between 'operational' knowledge (skill) and 'support' knowledge (understanding; cf Clancey, 1982) plays a major role. Support knowledge allows for generalisations, etc, while operational knowledge specifies what to do when to achieve a goal. The DIDACTIC GOALS are the basis for expanding the current knowledge of the user of the system and can be viewed as specifying the 'global needs' of the user.

(2) The second layer contains a (discourse) STRATEGY PLANNER. Planning consists of selection and top-down refinement of skeletal plans (cf. Friedland & Iwasaka, 1985) which represent typical didactic strategies or substrategies. Selection and refinement are based upon data in the LOCAL NEED. A local need identifies the immediate cause (e.g. question type, error, occasion for expanding the user's knowledge, etc.) and the assumed underlying cause (lack of some knowledge; misconception) of some

user action. For a further description see chapter 5.

(3) The third layer contains the TACTICS. TACTICS are the terminal elements ('executable methods') of strategies. They are elementary communication actions, consisting of a communication act (e.g. "To give you an example"), embedding a proposition that is the content of a TACTIC (e.g. "2dd deletes two lines from the current cursor position"). TACTICS are the input for a simple natural language generator (UTTERANCE GENERATOR, cf Stausholm, this volume), which takes care of lexical, syntactic and text-semantic issues like pronominalisation and ellipsis. In fact, the output of the COACH which is some sequence of tactics specifies already major pragmatic aspects of the discourse.

An overview of the architecture of the COACH is presented in figure 2. The components of the Coach are in capitals.

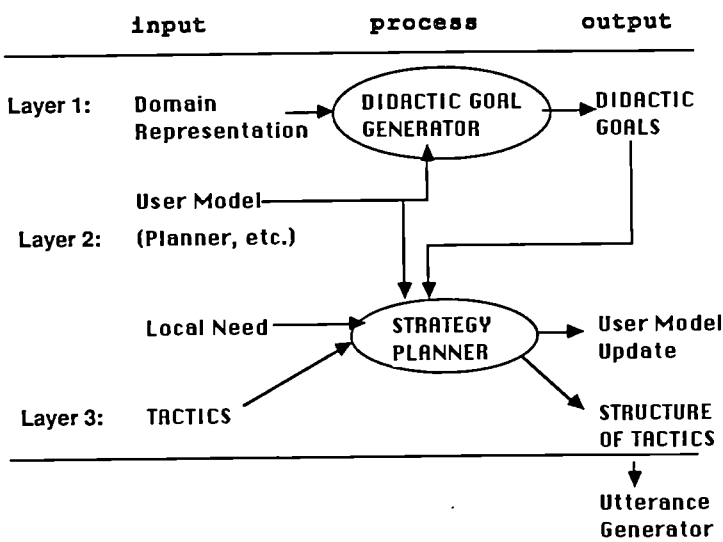


Figure 2 : *Input, Output, and Processing of the Coach*

4. Research Methodology.

Research has taken 2 parallel routes. An empirical route and a model construction route. The former tries to identify in a bottom-up way how human coaches plan their strategies and what tactics are employed. The model construction approach is top-down: it is aimed at the development of prescriptive notions about effective coaching, i.e. it specifies '(psycho)logical' concepts, derived from notions about optimising knowledge acquisition (cf. DIDACTIC GOAL generation). Model construction consists of designing and implementing a domain independent COACH which fed with a local need and access to knowledge structures (domain representation, user model, performance and coaching histories, current state of the IPS) constructs a strategy, which bottoms out in a sequence of tactics.

The two approaches complement one another. The empirical data keep the model 'honest' and 'ecologically valid'. The model construction not only provides an interpretation framework for the empirical data, but also 'criticizes' these data, in the sense that human coaches are not necessarily behaving in an optimal way for two reasons. The first reason is time constraints. The on-line coaching does not allow the coaches to carefully plan their actions. There is often backtracking. This leads us to the second reason: the coaches are not expert in coaching-by-teletype. They are limited in their 'normal' way to express their explanations. Moreover, there are large individual differences between coaches, and we want to have some semi-external criterion to select styles and strategies that appear to work. This can partially be abstracted from the empirical data, because the users provide thinking aloud protocols, which show in which way the actions of the coach are understood. The 'ideal model' is another framework to evaluate the usefulness of these empirical data.

Various empirical studies have been conducted (e.g. Bison & van der Pal, 1985 on Unix Mail; Sandberg, Winkels & Breuker, 1986; Hartley & Pilkington, forthcoming, on Unix Vi) leading particularly to a refinement and modelling of TACTICS. A number of typical coaching strategies have been identified, most of which focus on correcting and expanding operational knowledge.

The model construction has lead to an initial design and implementation of the STRATEGY PLANNER in Prolog (Winkels, 1987). A new version in Lisp is under construction, which will be integrated in the shell and empirically tested both against data from the empirical studies (see below) and from evaluations of an experimental help system for Unix Vi. Below we will present a more detailed description of the discourse STRATEGY PLANNER of the COACH.

5. Discourse strategies and strategy planning.

The selection and refinement of strategies is based on the local need. The local need consists of parts which specify different types of knowledge elements. The 'immediate cause' part in general contains a subset of the most recent performance history, i.e. the (sequence) of commands that caused an error, or occasion for expansion, feed back, etc. The 'diagnosis' part contains the specific lack of knowledge or a description of a misconception in terms of the domain representation.

First it is decided whether the local need should lead to coaching or to pure help. The difference is that the latter provides only ad hoc information, which is assumed not to be remembered, because -according to the DIDACTIC GOALS- the necessary conditions for expanding this particular knowledge are not present. Therefore, help is a stripped version of coaching. Next, a top level strategy is selected to tackle this local need. Top strategies are: remedial, expansion, state feed back. Each of these strategies can at some lower level be inserted in a strategy, i.e. function as a substrategy. For instance, many remedials imply an expansion of the knowledge of a user.
The top level strategy choice is based on the type of local need, and corresponds to the three major functions of a Help system:

Type of local need	**Function**	**Strategy**
error	remediate	remedial
occasion for expansion	expand knowledge	expansion
lack of feedback	provide feedback	state feed back

Once a top level strategy has been chosen, a selection and refinement process starts: If the local need is one that has been tackled successfully before by use of a specific plan this plan may be stored in a library of *skeletal strategies* (cf. Friedland & Iwasaki's "skeletal plans", 1985). If so, this skeleton will have to be instantiated to the current situation, and the planning is finished. In other cases, a more general plan is chosen, and will be refined hierarchically by applying refinement rules until the subgoals of the didactic plan are directly met by tactics (see 3.3). Conditions of these refinement rules may refer to the local need, the user model, the didactic goals, the current state of the system, and the state of the coaching dialogue thus far.
An example of such a rule is the following:

```
IF error (executable) & error (seriousness (high)) THEN
    insert-next-to (strategy (repair), strategy (context))
```

which means that after the context-strategy (which, for instance explains the immediate cause of an error) a repair strategy is inserted. A repair strategy consists of a plan to undo the effects of an executable error. In case the error is a serious one, the user may worry about undoing its effects, so the repair is provided first; in case the error is not a very serious one such a repair strategy is better delayed until the user has a full understanding of the (immediate) cause of the error.

The selected and refined strategies are in general comprehensive ones, which may have to be pruned by deleting branches containing information that is 'obvious' to the user, or -in case of pure help- that overloads the user with (new) information. An example of a simple strategy is presented in figure 3. The local need is of type error because the user accidentally typed 'p', which is a command he is not supposed to know and has never used before. The error is executable - the delete buffer gets inserted in the text - and considered moderately serious. Therefore the top level strategy chosen is REMEDIAL. The final tactic structure is presented in the first column of figure 3. The next column shows the text produced by the planner

which can be compared to that of one of our human tutors in reaction to the same error during an experimental session.

The first local need is:

```
local_need(error(executable, seriousness(moderate)), [performance_history([p])],
  [diagnosis(lack_of_knowledge, topic(concept(p), support(unknown),
  operational(unknown), [main_effect(p)])])].
```

Tactic structure	Text	Human Tutor
remedial(local_need6)		
clarify		
announcement		
drawing_attention	May I have your attention	
[interruption(prop22)]	please.	
context		*You used a particular*
clarification(perf_hist6)	You just did [p], that is what	*command, namely the p*
[instantiation(prop23)]	went wrong.	*command.*
new_information		
describe(topic11)		
describe_support(topic11)		
information1(topic11)		
[description(prop24)]	"p" is put buffer in text.	*that puts back the*
describe_operational(topic11)	Practically "p" means what	*last piece of text*
concretion(topic11)	you deleted last will be	*you have deleted before*
[operationalisation(prop25)]	inserted in the text.	*the cursor.*
state_feedback		*Hence you get back here*
describe_state_change(topic11)		*the line you just deleted*
check_assumption		*with dd.*
[elicitation(prop26)]	Is this you intention?	*Was that your intention?*
		If not, what then?
	|: no.	*> It was not intended.*
		Everything has to be removed.
repair		
describe(topic12)		*You can just place the*
information2(topic12)		*cursor on the line*
[direction(topic12)]	To undo do [crsr_up, dd].	*and do dd.*

Figure 3: Output of Strategy Planner and Human tutors.

The output provided by the planner has been compared to the tutoring texts provided by human tutors for about sixty different local needs. This comparison revealed first a remarkable agreement between the planner and the tutors. Secondly, the output of the planner is often 'better' in

the sense that new information is more explicitly linked to prior knowledge and no superfluous information is presented. Human tutors are inclined to give pure help, even when there is an occasion to expand the user's knowledge. And human tutors do often present information that is not applicable at that time. This last kind of information is easily forgotten by the user. Although the planner functions reasonably some aspects have to be developed further, in particular the management of simple dialogue.

6. Conclusions

The results described in the previous sections may appear natural and for all practical purposes adequate. However, the design and construction of the DISCOURSE PLANNER didn't follow a smooth path. When our investigations on coaching strategies started about 2 years ago, there was ample recognition of the fact that such strategies were hard or impossible to identify. Educational research had little to offer (cf. Ohlsson, 1986); the state of the art in constructing intelligent teaching systems consisted of fixed solutions * ; empirical data of human tutorial dialogues -as we have collected these ourselves- showed an almost endless variety, which lent itself to an almost similar variety of interpretations. It appeared that coaching is not some fixed expertise, where for each problem there is a ready made solution, as in analytic problem solving (Clancey, 1985), but coaching requires much more flexibility and rather consists of the synthesis (planning) of solutions. This has been recognised recently and there is an important trend in designing coaching systems which are planning systems (Elsom-Cook, 1987, Macmillan, 1987, Woolf & Murray, 1987).

Although this approach appears to be viable and leads to practical results, the theoretical basis is still not sufficiently established yet to allow for *generic* solutions. Coaching strategies are a subset of discourse strategies. Research in human discourse has a long and respectable tradition (e.g. rhetorics), but only in the last decade theoretical frameworks have emerged which focus on the selection and sequencing of the topics of discourse, i.e. the 'what' and 'when' to say things, which is preliminary to syntax and discourse semantics. The principles of topic selection and sequencing are probably not uniform. Some appear to be related to the semantics of the domain of discourse (e.g. McKeown, 1985); others are based upon the information management required for human serial processing (e.g. Reichman, 1981, Clark & Havilland, 1977). Besides many other issues for further investigation in the EUROHELP project, establishing the relations between these principles of didactic discourse planning is not only of theoretical interest, but may have many practical consequences, such as the construction of a 'multi-purpose' didactic discourse generator.

References

Bison, P., & van der Pal, F., (1985). Using UNIX-mail: an experiment in on-line tutoring. University of Amsterdam, Department of Social Science Informatics.

Breuker, J.A.P., (1987). Coaching in Help Systems. In: J. Self (Ed), *Intelligent Computer-Aided*

* With the notable exception of the design of 'Socratic dialogues' (Stevens & Collins, 1977).

Instruction. London: Chapman & Hall (in press).

Breuker, J.A.P., Winkels, R.G.F., & Sandberg, J.A.C., (1987). A shell for intelligent help systems. Paper accepted for proceedings of the IJCAI-conference, 23-28 August, 1987 (Science Track).

Clancey, W.J., (1982). Tutoring rules for guiding a case method dialogue. In: D. Sleeman E, and J.S. Brown (eds), *Intelligent Tutoring Systems*. New York: Academic Press.

Clancey, W.J. (1985). Heuristic classification. *Artificial Intelligence , 27* 215-251.

Clark, H.H., & Havilland, S.E., (1979). Comprehension and the Given-New Contract. In: R.O. Freedle (ed), *Discourse Processes: advances in research and theory*. Norwoord, NJ, Ablex.

Elsom-Cook, M., (1987). Discourse for Tutoring. Paper presented at the Third International Conference on Artificial Intelligence and Education, 8-10 May 1987, Pittsburgh.

Fischer, G., Lemke, A., & Schwab, T., (1985). Knowledge based help systems. In: *Proc. CHI 85: Human factors in computing systems*, ACM, New York, pp. 161-167.

Friedland, P.E., & Iwasaki, Y., (1985). The concept and implementation of skeletal plans. *Journal of Automated Reasoning, I*, 161-208.

Goldstein, I.P., (1982). The Genetic Graph: a representation for the evolution of procedural knowledge. In: D. Sleeman, and J.S. Brown (eds), *Intelligent Tutoring Systems*. New York: Academic Press.

Hartley, R., & Pilkington, R. (1987). Report on question answering in a help system, forthcoming.

McKeown, K.R., (1985). Discourse Strategies for Generating Natural-Language Text. *Artificial Intelligence, 27*, 1-41.

McMillan, S.A., (1987). Dynamic Instructional Planning: Global Planning. Paper presented at the Third International Conference on Artificial Intelligence and Education, 8-10 May 1987, Pittsburgh.

Ohlsson, S., (1985). Some principles of intelligent tutoring. University of Pittsburgh. To appear in: *Instructional Science*.

Reichman, R. (1981). Plain Speaking: A theory and Grammar of Spontaneous Discourse. Doctoral dissertation. Bolt Beranek and Newman inc.

Sandberg, J.A.C., Winkels, R.F.G., & Breuker, J.A.P., (1986). Coaching strategies and tactics of Unix-Vi consultants. UAM/EUROHELP/05. University of Amsterdam (The Netherlands).

Self, J., (ed), (in press). *Intelligent Computer-Aided Instruction*. London: Chapman and Hall.

Sleeman, D., & Brown, J.S., (eds), (1982). *Intelligent Tutoring Systems*. New York: Academic press.

Stevens, A., & Collins, A., (1977). The goal structure of a socratic tutor. In proceedings of the Association for Computing Machinery Annual Conference. (Also available as BBN Report No. 3518 from Bolt Beranek and Newman Inc., Cambridge, Mass., 02138).

Woolf, B., & McDonald, D.D., (1984). Context dependent transitions in tutoring discourse. *Proceedings of AAAI 1984*. Los Altos: Kaufmann.

Woolf, B., and Murray, T., (1987). A Framework for Representing Tutorial Discourse. Paper accepted for proceedings of the IJCAI-Conference, Milano, 23-28 August, 1987.

Project No. 857

M A R G R E T - A PRE-PROTOTYPE OF AN 'INTELLIGENT' PROCESS MONITORING SYSTEM

P. Elzer, H. W. Borchers, H. Siebert, C. Weisang, K. Zinser

Brown Boveri & Cie
Central Research Laboratory
ZFL/L3
POB 101332
D-6900 Heidelberg 1
F.R.G.

1. INTRODUCTION

In the ESPRIT-Project P857 'GRADIENT' new techniques for Man-Machine Dialogue and Operator support in dynamic systems are developed and/or evaluated. The aims and general principles of this project have already been described several times, e.g. in /1/. Since the start of the project in October 1985 two years have now passed and several of its goals have been reached. The respective results are described in separate papers, e.g. /2/, /3/, /4/. This particular paper deals with a successful experiment concerning the integration aspects of a GRADIENT-System. The approach chosen was to construct a pre-prototype of the entire system, realizing the essential components with limited functionality, using a powerful development tool. This method conforms to the meanwhile established procedure for the development of knowledge-based systems. The results so far have fully justified the effort.

In particular it was regarded necessary to gain experience with the following aspects of a GRADIENT system:

- Applicability and user acceptance of modern and alternative presentation techniques in industrial control, like information zoom, multi-windowing, panning etc.

- Possibilities and user acceptance of new operator input techniques like soft-keys, mouse, sensitive areas, etc.

- Details concerning the functionality of support tools for the preparation and design of picture contents and dialogue sequences, especially the GRADIENT 'Intelligent Graphical Editor' series.

- Methods and problems concerning the connection of knowledge based systems in realtime to dynamic systems.

- Methods and problems of cooperation and data exchange among knowledge based systems and between these and other computational processes, especially with respect to distributed systems and the definition of the 'composite objects' within GRADIENT.

- Capabilities and limitations of commercially available support software and 'A.I.-oriented' hardware.

The pre-prototype was developed at the BBC Central Research Laboratory in Heidelberg, F.R.G. For easier reference it was named 'MARGRET' (Multifunctional All-puRpose GRadient Experimental Test Environment).

It will be used as a vehicle for test, integration and demonstration for the whole duration of the GRADIENT project. This of course implies that it will have to be upgraded and structurally improved in regular intervals according to experience with its use and new technical insights.

2. ARCHITECTURE OF MARGRET

2.1. The GRADIENT System

The GRADIENT System as a whole is intended to demonstrate qualitative improvements in a wide variety of rather complex subtasks in S & C of dynamic processes. These subtasks can be described as intelligent fault diagnosis support (Quick Response Expert System (QRES) and Support Expert System (SES)) and intelligent graphical interfacing of process information (Graphical Expert System (GES) and Presentation System (PS)). Fig. 2.1 illustrates the interrelations between these components.

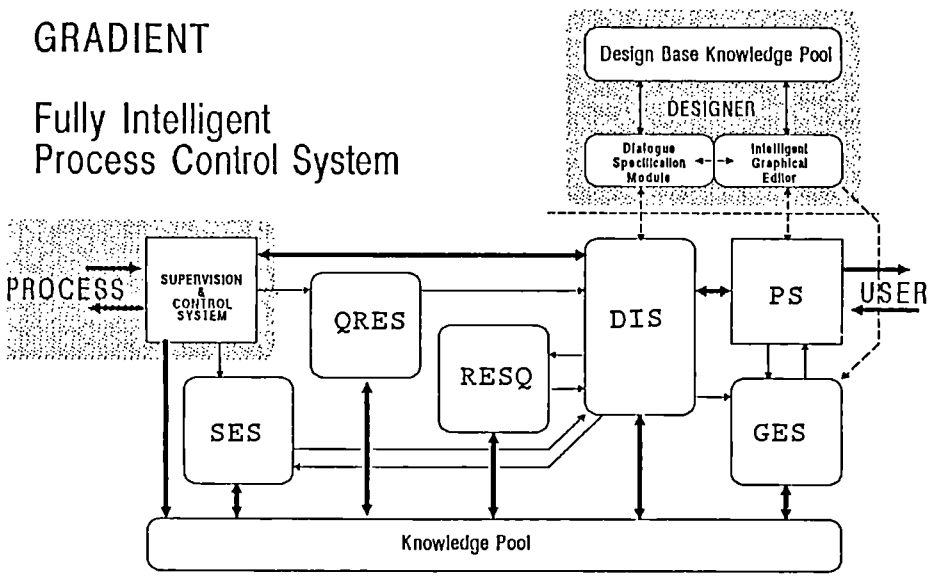


Fig. 2.1: The Structure of the GRADIENT System

2.2. The MARGRET Preprototype

2.2.1. Overview

The pre-prototype consists of models of the following important components of the GRADIENT project:

- Quick Response Expert System
- Dialogue System
- Graphical Expert System
- Presentation System

Figure 2.2 presents a schematic structure of the pre-prototype with the above-mentioned components, drawn as rectangles, and the data exchanged between them, drawn as circles. According to the terminology developed within the GRADIENT consortium these data are called 'Composite Objects' ('CO's').

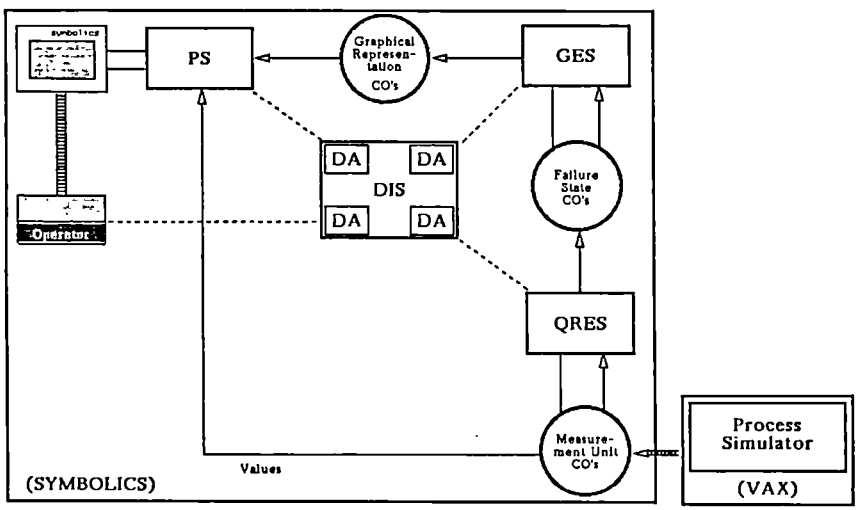


Fig. 2.2: The Internal Structure of the MARGRET Pre-Prototype

In the next phase of the project parts of MARGRET will be ported to a SUN workstation in order to benefit from the graphical capabilities of this hardware. Then the data to be exchanged between modules residing on different machines will be encoded according to the 'Composite Objects' convention and sent to the receiver, which will decode the data again into its internal representation.

2.2.2. Implementation

MARGRET has been implemented in KEE3 on a Symbolics 3640 running ZetaLisp under Symbolics' operating system 'Genera 6.1'. MARGRET relies on the use of KEEpictures, RuleSystem3 and Symbolics' specific process handling mechanisms. These tools do not yet guarantee a totally realistic system behavior, especially with respect to response time, but have proven to be valuable for rapid prototyping.

The process simulator models a conventional power plant. It is written in FORTRAN and runs on a MicroVAX II under VAX/VMS. For the time being, the simulator submits a set of 26 analog values describing the state of the power plant. Every five seconds an updated set of simulated process values is sent from the MicroVAX II via Ethernet (under the DECnet protocol) to the Symbolics.

2.2.3. Functionality

The functionality of MARGRET's components is best described in terms of their interaction. With reference to Figure 2.2 the following list illustrates a sequence of actions in MARGRET:

- Data values are read asynchronously from the process simulation on a VAX into 'Measurement Unit CO's', and also transmitted to the PS directly.

- QRES reads 'Measurement Unit CO's', and
  - creates 'Failure State CO's', and
  - informs its Dialogue Assistant (DA).

- GES is informed by its Dialogue Assistant to read 'Failure State CO's'.

- GES creates 'Graphical Presentation CO's' and
  - displays appropriate messages directly, and
  - informs its Dialogue Assistant.

- DIS waits for Operator Actions, and (in case)
  - informs PS of selected actions through DA's.

- PS displays selected Graphical Representation CO's.

The synchronization of all modules of the GRADIENT system is in MARGRET mainly handled by the rudimentary implementation of the DIS module.

3. USER INTERFACE

Fig. 3.1 shows a complete screen layout with all tne windows which are available to the user of the MARGRET preprototype system, though they need not necessarily appear on the screen all at the same time.

3.1 The 'Process Monitor' window

The 'Process monitor' window is the operator's window to the process. It looks at a complete picture of the plant. The operator can 'move' through that picture and cause the system to display more details of the plant by zooming in. The picture is organized to support 'additive' as well as 'alternative' 'information zoom'.

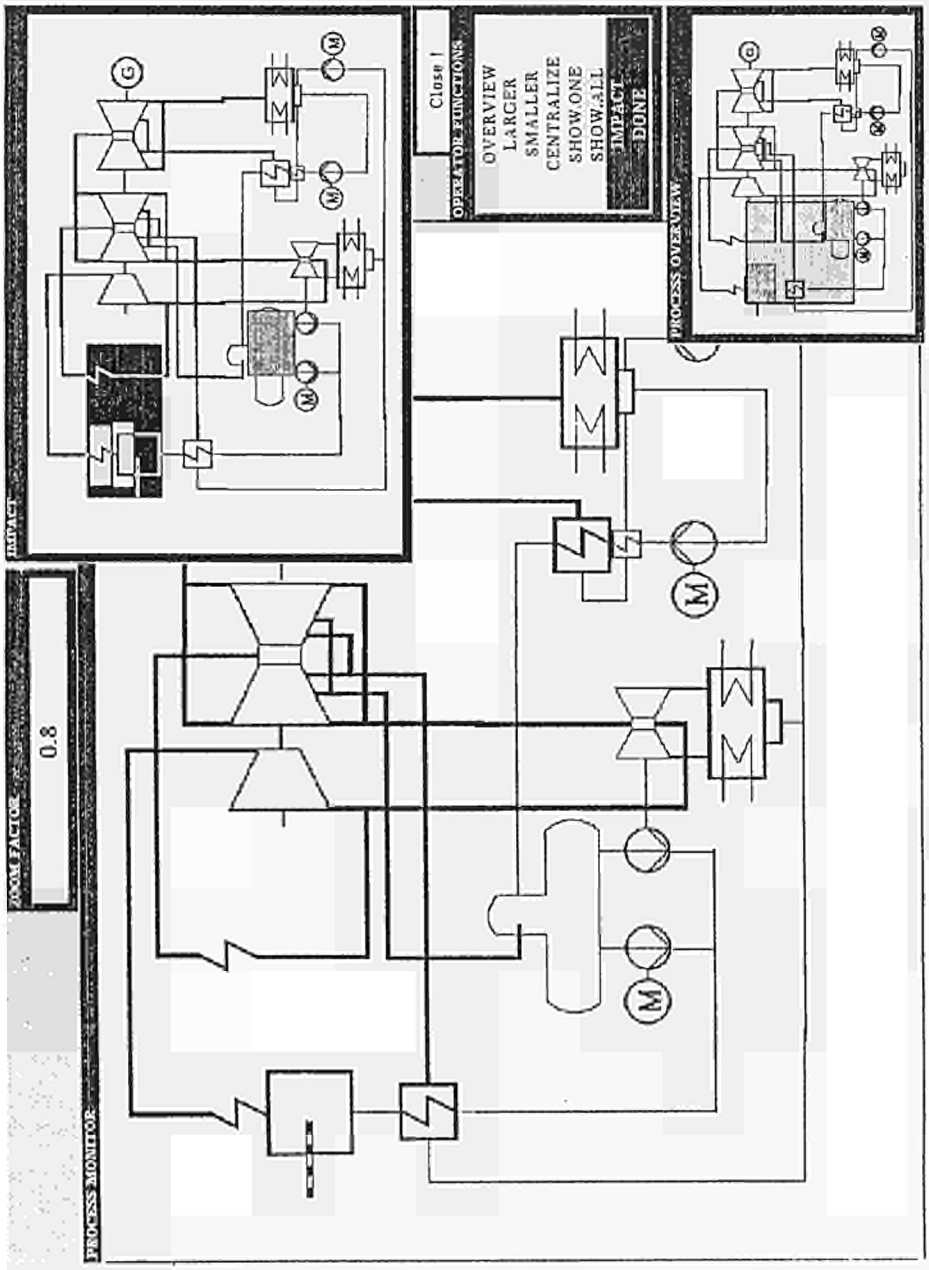


Fig. 3.1: The Complete Set of Windows in MARGRET

'Additive information zoom' means that more details are added to the picture when its zoom factor exceeds some predefined limit. Figure 3.2 shows an example from the feedwater system. In this case the flow and temperature measurement values are added when the operator zooms in.

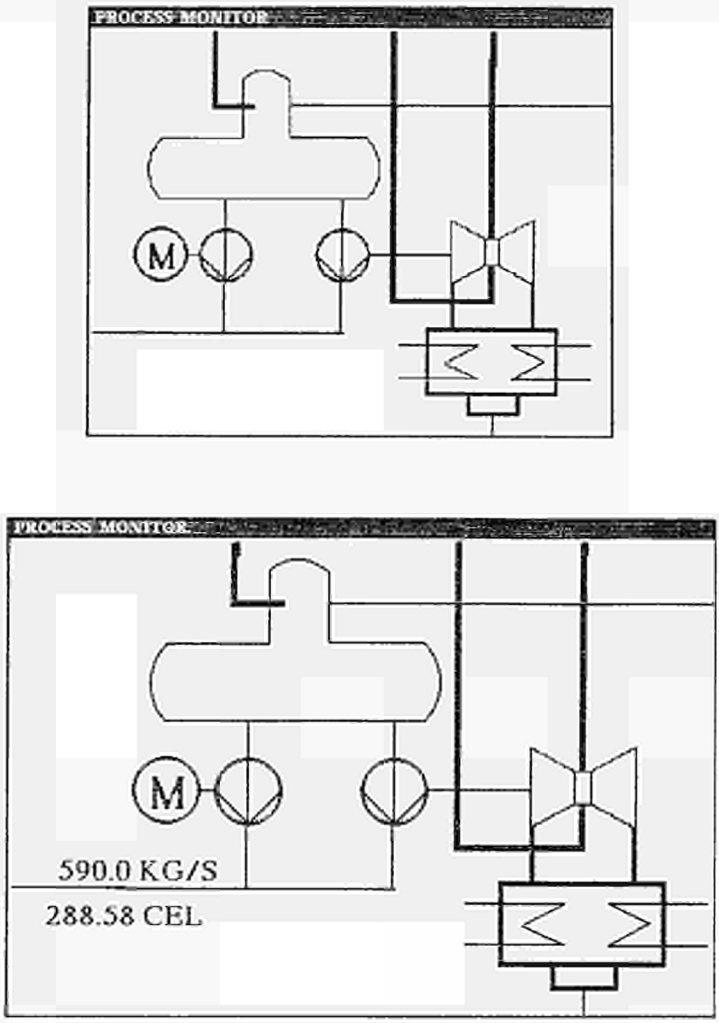
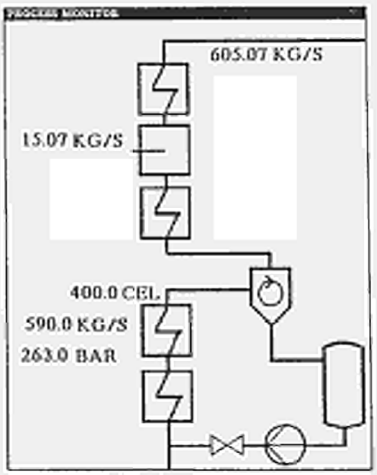


Fig. 3.2: Additive Information Zoom

'Alternative information zoom' (Fig. 3.3) means that the graphical representation e.g. of a subsystem is completely replaced by a more detailed representation when the zoom factor exceeds an appropriate predefined limit.



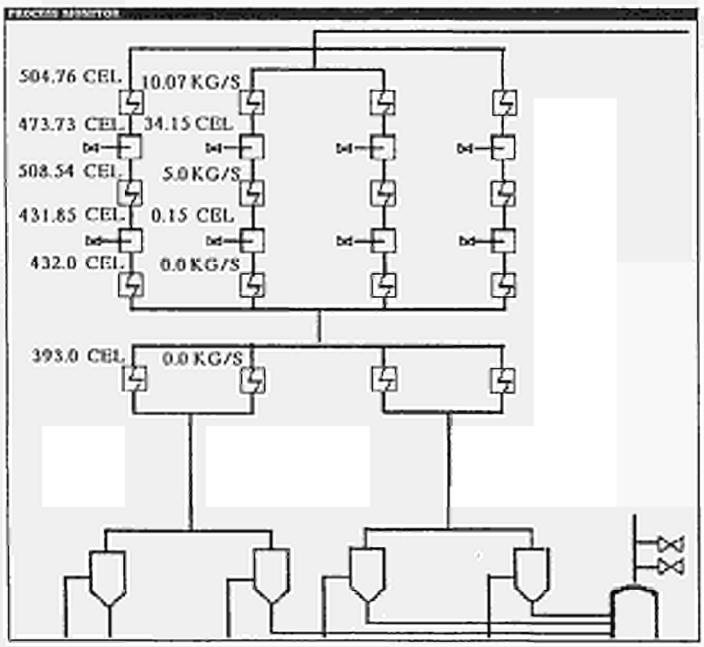


Fig. 3.3: Alternative Information Zoom:
Details from the Steam Generator System

3.2 'Process overview' window

This window at the bottom right of the screen is always present and contains an overview picture of the entire process. The failure states, which are discovered by the QRES (Quick Response Expert System), are signalled there by gray rectangles displayed at the appropriate position.

If there is more than one failure state detected, the system determines the smallest rectangle containing all the failure state rectangles and displays this rectangle within the 'Process overview' window in light gray, as shown in Fig. 3.4.

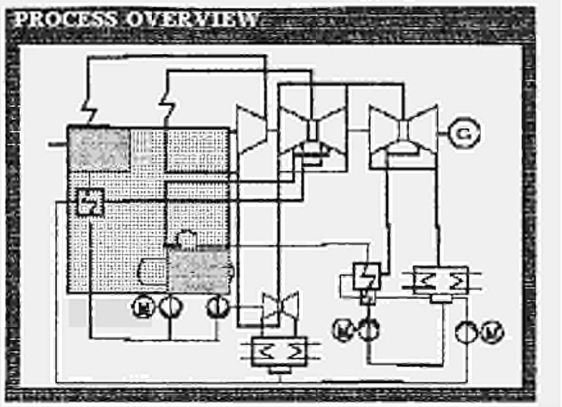


Fig. 3.4: Overview Window with Indication of Failure Syndromes

Each of these small rectangles represents at the same time the system's proposal to the user as to which portion of the plant picture should be looked at to retrieve further information on the failure state in question. The user can follow the system's suggestion by invoking the function 'SHOW.ONE' (cf. section 3.3 and Fig. 3.5).

If the operator invokes the function 'SHOW.ALL', the portion of the plant picture covered by this light gray rectangle will be displayed in the 'Process monitor' window using the smallest possible zoom factor. Fig. 3.6 shows the effect of this function.

3.3 'Operator functions' window

At the right margin of the screen there is a window containing eight subwindows titled 'OVERVIEW', 'CENTRALIZE', 'LARGER', 'SMALLER', 'SHOW.ONE', 'SHOW.ALL', 'IMPACT' and 'DONE'. These subwindows are mouse sensitive and serve as 'soft-keys' with the following functionality:

'OVERVIEW': An overview picture of the entire plant will appear in the 'process monitor' window.

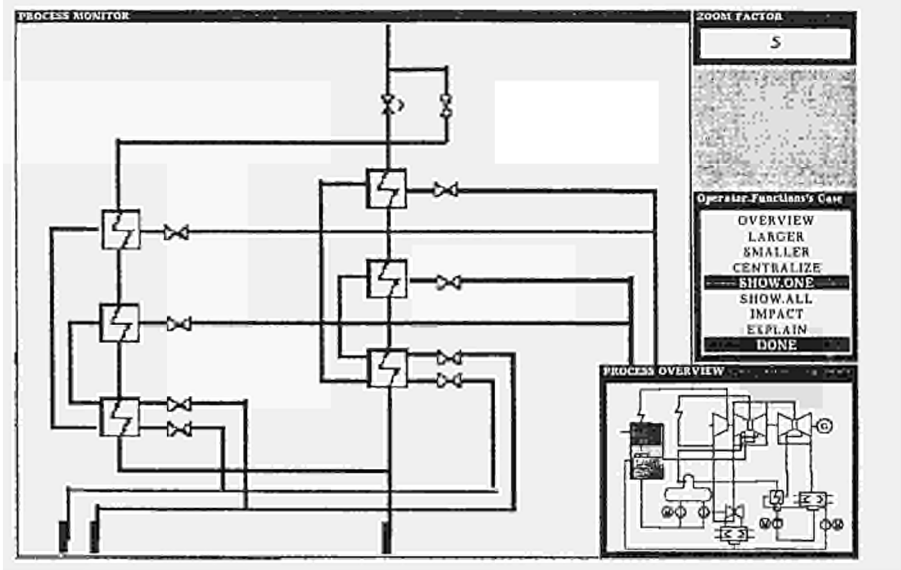


Fig. 3.5: The Effect of the 'SHOW.ONE' Function

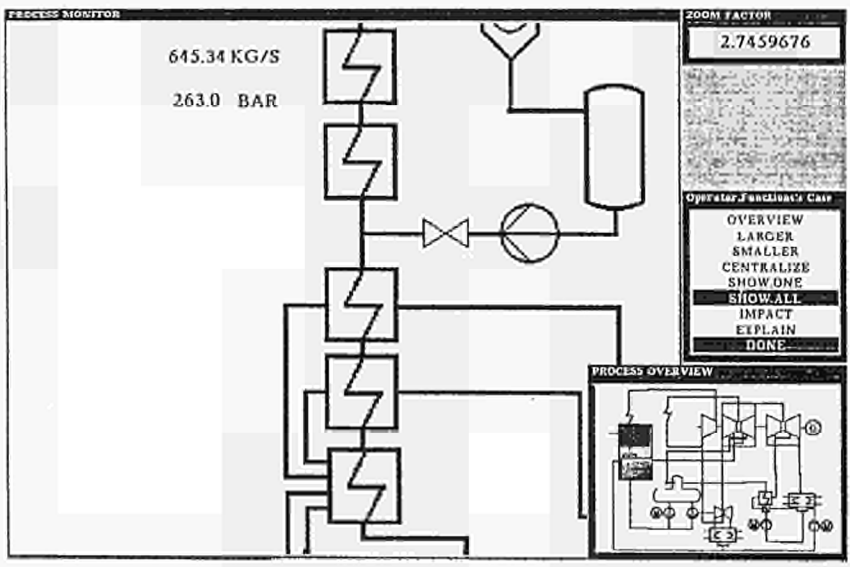


Fig. 3.6: The Effect of 'SHOW.ALL'

'CENTRALIZE': The component, which is selected by the operator (either in the operator's window or in the overview window) will be moved to the center of the 'process monitor' window.

'LARGER': Causes the picture in the 'process monitor' window to be enlarged by increasing the zoom factor one step.

'SMALLER': Reduces the zoom factor one step.

'SHOW.ONE': Causes information about one failure state which can be selected by the operator to be displayed in the 'process monitor' window (c.f. 3.2).

'SHOW.ALL': Causes information concerning the entire detected failure syndrome to be displayed in the 'process monitor' window (c.f. 3.2).

'IMPACT': Automatically opens an additional window and displays in it information about possible consequences of a failure situation.

'DONE': Indicates that one of the functions has been carried out.

3.4 'Impact' window

This window is not necessarily open all the time. It will be opened automatically by the system and can be placed by the user at any position on the screen, after 'IMPACT' has been activated (cf. also section 3.3). The operator can select one of the active failure states by clicking on the according gray rectangle in the 'Process overview' window. This rectangle will then be repeated in the 'Impact window', and the subsystems that might suffer from the failure state in question are indicated by displaying them in 'reverse video' in the 'Impact window'. Figure 3.1 e.g. shows the impact of a failure state in the feedwater system.

3.5 'Zoom factor' window

This window shows the currently applied zoom factor as a positive number. It has turned out to be a valuable aid for the operator indicating his position in the 'scale dimension'.

4. KNOWLEDGE-BASED COMPONENTS WITHIN 'MARGRET'

When it was decided to implement a preprototype of the GRADIENT system, it was clear that not all of its knowledge based components could be implemented as expert systems.

For constructing the pictures representing the process a simple, object oriented graphical editor was used which was part of the development tool set. The graphical objects produced by this editor can be easily accessed by reasoning processes without any interface problems.

The Support Expert System, which will enter the GRADIENT system only later, was completely dispensed with. The Dialogue System was substituted by a synchronising mechanism, because the DIS is being developed at another partner's site (University of Strathclyde, Glasgow) and no prototype was available at the time of development of MARGRET.

But at least two knowledge based components are necessary to keep the prototype running, one to detect failure states from (abnormal) process values, the QRES, the other to display information about known failure states, the GES. Of these, only QRES was implemented as a rule-based expert system because:

a) there was inhouse knowledge available concerning failure states of power plants, and

b) the expert system development tool KEE at the moment does not yet support two reasoning processes running in parallel on one machine.

Therefore, the operations of the graphical systems were implemented as LISP functions in an object oriented programming style.

For purposes of the preprototype, GES was designed as a set of graphical operations. Some of the operations are running in sequence every time a new failure state has been detected by the system or an old failure state has been recognized as not being valid any more. Others are triggered by the operator interactively. A description of the effect of these operator functions has been given in section 3.

The QRES has been implemented in a pretty straightforward manner. It contains only a few rules, covering mainly failure states in the feedwater system.

At a later stage it will be replaced by an adaptation of the QRES development at CRI.

5. CONCLUSIONS

Although MARGRET originally had only been planned for experiments with graphical aspects of GRADIENT, it turned out to be a highly valuable source of experience in other areas as well. Two of these shall be briefly mentioned here:

a) Input to the discussion about 'Composite Objects' within GRADIENT. As the components of GRADIENT are developed at different sites and will run on different computers it has turned out to be necessary to define a uniform data exchange mechanism between these components. The best method seems to be the exchange of standardized objects which contain all the necessary information to be transmitted between GRADIENT components or subsystems. Within MARGRET, the following types of composite objects were tried out (c.f. 2.2.3):

   - Measurement Unit CO's
   - Failure State CO's
   - Graphical Representation CO's

b) Synchronization and Scheduling in an A.I.-Tool based environment. Synchronization and Scheduling of parallel processes are of prime importance in realtime systems, but little experience exists as yet concerning their implementation and application in an 'A.I.-typical' tool environment. However, in order to come to a really working MARGRET system, it was necessary to implement the synchronization and scheduling mechanisms with a realistic functionality. The result of this work provided valuable and decisive insights into the possibilities of the underlying hardware and support software and will definitely influence the future design of GRADIENT.

Of course a lot of problems have still to be solved in order to arrive at a fully functional GRADIENT system like e.g.:

- Computer-aided development of the large overview pictures within MARGRET (which is one of the tasks of the 'Intelligent Graphical Editor', to be developed within the GRADIENT project),

- Inclusion of colour and improvement of speed,

- True parallelism of rule based systems etc.

But the results so far have confirmed our confidence in the achievability of these goals. After all, to our knowledge MARGRET is the first system in Europe with this particular functionality and all demonstrations to potential users have indicated that it would be regarded as a significant improvement over the current state-of-the-art.

REFERENCES

/1/ Alty,J., Elzer,P., Holst,O., Johannsen,G., Savory,S., Smart,G.: Literature and User Survey of Issues related to Man-Machine Interfaces for Supervision and Control Systems; ETW 1985, pp.719-729

/2/ Christensen,B.: The Development of QRES - A Real Time Experrt System for Process Monitoring; paper submitted for ETW 1987

/3/ Hollnagel,E., Sundstroem,G., Weir,G.: User Modelling in GRADIENT; paper submitted for ETW 1987

/4/ Alty,J., Weir,G.: Dialogue Design for Dynamic Systems; paper submitted for ETW 1987

Project No. 820

A TOOLKIT FOR BUILDING KBS APPLICATIONS FOR PROCESS CONTROL. FIRST ACHIEVEMENTS OF THE PROJECT.

Alberto Stefanini (CISE)
Francois Arlabosse (FTC)
Alfonso Cavanna (ANS)
Pierre Courtin (AER)
Mogens Levin (FLS)
Roy Leitch (HWU)
Pierre Martin (CAP)

(CISE) CISE s.p.a., Milano, (I)
(FTC) Framentec, Paris, (F)
(ANS) Ansaldo, Genova, (I)
(AER) Aerospatiale, Cannes La Bocca, (F)
(FLS) F.L. Smidth & Co. A/S, Valby, Copenhagen, (D)
(HWU) Heriot Watt University, Edinburgh, (GB)
(CAP) CAP Sogeti Innovation, ZIRST, Meylan, (F)

ABSTRACT

Project 820 is aimed at prototyping and experimenting a KBS development environment for applications in the domain of industrial control. During the first year of work, the basic development environment (termed the 'kernel toolkit') has been specified and is currently being implemented. It takes the form of a toolkit, i.e. a set of modular, integrated tools plus a design methodology.
As two basic approaches exist for modelling a system to be controlled, i.e.:

- empirical: based on relationships expressing observed associations;

- ontological: based on relationships that derive from a theoretical understanding of the domain, and following the approach of qualitative modeling,

the kernel toolkit provides knowledge engineering tools suitable for both these approaches to system modelling.
Development of the toolkit takes place in parallel with three large demonstrator applications (relevant to control of a power plant, a satellite, and a cement manufacturing plant) where the tools will be experimented and validated.
This paper provides motivations for the main design choices of the toolkit as well as an overview of the most important features of the three demonstrators.

1. INTRODUCTION

Among the most demanding areas where expert systems are expected to play a major role within a few years, process control is attracting a significant interest. Expert systems in process control environments appear of particular importance for predicting faults in technical and chemical processes, as well as for providing guidance and planning for actions meant to repair or prevent faults. In the frame of ESPRIT an exploratory study contract on the subject was funded in 1985 (Project P256: Time Dependency and System Modeling in Knowledge-Based System Design for Process Applications). The study focused on a set of requirements for a knowledge-based system (KBS) development environment for process control applications. It examined a set of new techniques for knowledge representation and inference, whose suitability was tested on a sample of case studies, and outlined a reference architecture of the KBS development environment. The actual development of the environment is the subject of a further and larger ESPRIT project, whose contract has been awarded in the 1986 group of projects in the area "Advanced Information Processing", reference number P820.

2. THE TOOLKIT CONCEPT

The large spectrum and specific nature of the envisaged application area, and the different approaches to knowledge representation we will overview hereinafter, are motivating our main design choice, i.e. to structure the KBS building environment as a toolkit comprising a design methodology and a set of special-purpose tailorable building modules. With respect to traditional KBS shells and building tools, the toolkit approach offers a much more flexible and specialised working environment, enabling the KBS designer to build his own target application by selecting from the toolkit the appropriate modules, and by tailoring and composing these to meet individual requirements.

A toolkit comprises:

* A general system architecture that will serve as a common reference framework. The architecture specifies the basic structure and function of the main component modules. These include: knowledge bases, external interfaces, working memories and inference mechanisms. The architecture specifies how the modules interact and defines the flow of communication information between them.

* A set of special-purpose building modules, that can be tailored and assembled together to construct the target KBS building tool. This set will include all modules necessary for the implementation of the inference mechanisms and special facilities (e.g., explanation and justification) specified in the reference architecture.

* A set of engineering support tools to aid the system designer in the construction of the knowledge bases (i.e., editing, debugging, refining, and updating the represented knowledge).

* A set of specifications for the construction of external interfaces towards traditional programming environments

(e.g. DBMS, simulation packages etc.) and towards the industrial process (data acquisition, sensors, instrumentation actuators etc.).

* A KBS analysis and design methodology for supporting the correct and effective use of the toolkit over the range of tasks covered by the application domain.

The paradigm of the toolkit allows high flexibility to explore different architectures for structured and efficient knowledge organisations, and still retains the classical advantages of open-ness of the toolkit approach. The toolkit concept embodies a good compromise between the contrasting exigences of generality and specific, effective usability.

2.1. Main approaches to system modeling

The central motivation for this project is the firm conviction that the construction of an artificial system capable of interacting with the complex dynamic processes found in industrial environments requires a sophisticated blending of empirical (inductive) and analytic (deductive) techniques.

Two main approaches to system modelling can be identified. One, currently being developed in the KBS area, takes the world as we find it, examines the various systems that exist in it and then infers relationships and regularities about the observations that can be made. This method is essentially empirical and intuitive. It has the basic advantage that it remains close to reality and that the models are readily accessible and understood by people. On the other hand this approach lacks mathematical elegance and deductive strength, and can appear little systematic. Models obtained by empirical observation are specialised, and often lack generality. A second method, the deductive approach, takes the opposite view. Instead of studying a particular system, it goes to the other extreme and considers the set of all conceivable systems with common properties, usually in the form of fundamental physical principles. Equivalence classes are thus formed and used to classify a particular system. Once a system is assumed to belong to a given class, it inherits the properties of that class. Both approaches have their relative advantages.

We propose a novel architecture capable of supporting both of these approaches to system modelling: an empirical model which uses established knowledge from observation of the systems behaviour and an ontological (analytic) model based on the application of general physical laws to obtain an explicit qualitative model of the process.

Empirical models consist of relationships, often in the form of condition-action pairs, that have been acquired through first-hand experience with the system at hand, or from a more experienced expert. These relationships allow inferences to be made in a given situation, but do not provide any kind of inherent reason for the relationship itself. These relationships are shortcuts through, or partial compilations of, a deeper understanding of the problem domain. They tend to be experiential and model the decision making process rather than the process itself. However, when applicable, such knowledge is extremely efficient.

Ontological models, on the other hand, provide the detailed causal, functional and physical information about the operation of the system. Such knowledge is essential for coping with new situations or for developing diagnostic systems based upon the internal structure of the system. Ontological knowledge is, however, more complex to represent and requires sophisticated inference procedures. The goal of our project is to combine the speed and efficiency of empirical knowledge with the generality of ontological knowledge obtained from 'first principles'. This requires a co-ordination and control strategy such that, when applicable, the empirical model is used, resorting to the ontological model only when required.

The above considerations determine a toolkit architecture consisting of three sets of tools: for expressing and using ontological models, for expressing and using empirical models, and for structuring communication between, and control over, different models.

Ontological models require a representation language and associated inference mechanisms allowing:

a) the representation of the structure of the system in terms of fundamental primitives and the constraints between them. A number of alternative approaches exist to represent the structure of physical processes, essentially characterised by the basic unit of representation;

b) to determine the flow of causality in the structural model, i.e to generate a causal net expressing qualitative influences among the physical parameters;

c) to determine the dynamic behaviour of the system, i.e simulation of the evolution (over time) of the system. This is essential for the modelling of dynamic systems, and for reducing the ambiguity in the propagation of disturbances in static models of dynamic systems.

Empiric knowledge is mainly declarative and fragmentary in nature, and KBSs have achieved up to now major successes in those application domains where knowledge involved is empiric. In these domains, the rule-based paradigm typical of ESs has offered a basic advantage over traditional programming, in that it allows the programmer to tackle problem solving at a higher level of abstraction and in a more flexible and natural way. Many classes of problems however exhibit, in addition to empiric declarative knowledge, well structured chunks of procedural knowledge (still mainly of empiric nature) that should be represented and used in the form it is naturally available. Consider, for example, tasks such as decision making, fault detection and diagnosis, sensory data interpretation, monitoring of industrial processes, all relevant to the intended application area of the toolkit. In these cases, it is highly inconvenient, for many reasons, to disperse the available procedural knowledge into a flat declarative representation; it is thus necessary to introduce an adequate formalism for representing procedural knowledge, and an appropriate inference mechanism, allowing the combination of the structured coding of procedural knowledge with the declarative representation typical of a pure rule-based system. Therefore empirical models can require separate languages for representing both procedural and declarative associational relationships.

The overall operation of KBSs using different models of the controlled system, expressed by means of different representation paradigms, requires a sophisticated control and co-ordination framework performing the following functions:

management of inter-module communication, reflecting the existing dependency among different representations;

management of the interaction among different views of the system, expressing the viewpoint of a particular agent in charge of performing a particular task, with the aim of integrating their contribution to achieve a specified goal.

An additional issue posed by complex process control applications, where time and space constraints may impose a distributed computing architecture, is control of co-operative problem solving in a distributed environment.

3. INTENDED APPLICATION AREAS

The field of industrial process control is currently attracting much attention from KBS designers as an area offering significant potential for improved efficiency and safety of operation. This impetus stems from the recognition that many industrial control tasks remain outside of the scope of current automatic control methods. KBS technology provides the potential for extending automation to cover a wider range of tasks by using symbolic processing methods similar to those used by humans.

Typical tasks that can be envisaged for KBSs in the industrial process control environment include:

Situation assessment and fault detection:

Assessment of the operating conditions of a physical system, that is, complete (or partial) identification of the system state starting from incomplete and scattered data, taken from present and past observations (measurements). This task also includes identification of possible deviations from normal (desired) operating conditions, i.e. fault detection.

Prediction:

Evaluating the dynamic trend of a physical system, i.e. identifying a sequence of future system states, once the present state and input perturbations to the system are known. Essentially, this is a simulation task, but can also be used for fault propagation and for determining appropriate (feedback) control action.

Fault diagnosis:

The identification of the mechanisms which caused a difference between a (partially known) actual state and an expected state whose prediction was based on an explicit or implicit model of the process. At present this is the most significant task that currently lies outside of automatic control systems.

Regulation:

The manipulation of control variables (actuators) to alter the behaviour of a physical system in order to keep it close to desired one. The task here is to maintain a given equilibrium condition in face of system disturbances. Continuous feedback control is the main method used to achieve process regulation.

Tracking

The manipulation of control actuators in a way as to follow a prescribed state trajectory or path. This task arises during plant start-up or shut-down or during a significant change in the desired operating region, e.g. product throughput or different product campaigns. This task employs a mixture of continuous feedback control and sequential or supervisory control. This task shares many similarities with planning problems.

Some of the above tasks can already be adequately automated for many industrial processes. In particular, regulatory feedback control, generally known as process control, has a range of sophisticated theories that have shown many spectacular successes, and some lesser known failures. These methods are based on analytic representations of the process based on real-valued continuous or discrete time functions. Such methods require a precision in the specification that in many cases cannot be attained. These methods moreover seek a unique solution and require a complete specification of the process. Imprecision or incompleteness cannot be handled using these methods, although uncertainty, in the form of probability density functions can.

In the case of fault diagnosis, adequate analytic representations are extremely difficult to obtain. This is mainly due to the large variation in the variables (signals) being processed by the system during these tasks. Simplified 'small signal' models are not adequate for these tasks. In addition, for the fault diagnosis task, a fault may induce a change in the system structure requiring a significantly different model of the process.

A general disadvantage with the analytic methods is that they do not provide a human window through which the operator can observe the operation of the system. The mathematical description of the process is far from the mental models of the human operator. The provision of a human window is a defining attribute of the KBS approach. In industrial control this aspect is essential for human supervision and safety. Consequently, even those tasks that can presently be automated by traditional methods can benefit from a KBS approach, assuming that the same performance can be achieved.

The above inadequacies of traditional methods are motivating current efforts in Artificial Intelligence to establish methods of formal reasoning about system descriptions expressed in symbolic, or qualitative form. Such systems offer the potential of providing causal explanations of their behaviour in terms readily understood by a human. Even if a quantitative model is available and adequate for a particular process, there may also be a significant advantage in reasoning qualitatively about behaviour. This can bring computational benefits as well as cognitive ones. However, for many processes and tasks, quantitative data are just not reliable or available, in which case qualitative methods provide the only possible automated solution.

4. ANALYSIS OF THREE DEMONSTRATOR CASES

As previously mentioned, within the project three demonstrator applications in the area of the process control and supervision will be implemented. The demonstrator cases will focus on:

1. malfunction detection and diagnosis in the thermal cycle of a power plant;

2. data interpretation and control of a spacecraft, including misposition/attitude detection and diagnosis, and assistance to correction manoeuvres;

3. startup, on-line monitoring, and control of vital subsystems in a cement manufacturing plant.

4.1. The power plant case

The experimental application domain considered in the design of the first demonstrator is the conduction of a power plant. Among the several different tasks included in this application domain, we focus on the task of overall plant diagnosis. This task consists of the identification of malfunctions that affect major plant components, by evaluating plant efficiency and comparing it with known reference conditions. Plant efficiency evaluation is a very complex and knowledge intensive activity. As a matter of fact, it is not possible to rely upon the plant efficiency formula. Instead the actual plant performance is estimated by considering qualitative evaluations of a certain set of index variables correlated to the subsystems into which the plant is subdivided. Overall plant diagnosis includes two more primitive tasks: system state assessment and diagnosis.
The relevant knowledge comes from many different sources, ranging from heuristics owned by conduction and maintenance staff, to more formal technical knowledge owned by plant designers. Thus it includes empirical, incomplete and uncertain knowledge mixed with qualitative process models. Therefore this demonstrator will employ an ontological model of the plant together with declarative formalisms expressing the above mentioned empirical knowledge.

4.2. The spacecraft case

This application domain is concerned with supplying expert support to the operators in the control room of a geostationary satellite for telecommuncations. A telecommunication satellite in geostationary orbit, is expected to accomplish several missions. In case of attitude loss the main goal of the control station is to run the appropriate actions in order to address, as quickly as possible, the re-starting of the mission. One of the most important constraints underlying the attitude control actions, consists of the non-observability (or partial observability) of the internal and external satellite states. This is due essentially to the satellite's design and the limited number of available telemeasures. Actions and observations are thus strictly coupled.
The operator activities consist mainly in the detection of the satellite misposition/attitude, and in their correction through suitable correction procedures. These tasks are based on the interpretation of telemetry data coming from the satellite. In order to cope with datainterpretation and correction procedures

the demonstrator will mainly employ empiric procedural knowledge representation.

4.3. The cement manifacturing plant case

The third application domain concerns process control and on-line monitoring of a cement manufacturing plant. A cement manufacturing plant is constituted by several subsystems. The control task is very important for the plant because it requires continuous attention during normal plant operation. We have recognized three main tasks to be performed within this application:

- regulation of the kiln during operation;
- tracking during kiln start-up and shut-down;
- regulatory control of the mill;

These objectives involve several kind of knowledge. The kiln and mill control compell the human operators to take care of a large amount of signals in the control room. The discrimination over those signals is carried on by employing empirical knowledge. The start-up/shut-down of a kiln implies sets of sequential procedural activities to be performed in a parallel but synchronized way.
The demonstrator will mainly employ empirical models expressed by declarative formalisms in order to cope with the regulatory control task, while within the start-up/shut-down task it will employ empirical procedural formalisms to represent the complex sequences of actions it involves. Moreover, an ontological model of the kiln, representing well established theories owned by the plant designers, will be employed for prediction.

5. TOOLKIT SPECIFICATION

The toolkit provides representation languages for expressing both empirical and ontological models of the process. Three languages have been specified:

a) a Component-Based Language for representing ontological qualitative system models;

b) A Rule-Based Language for representing empirical declarative relations;

c) an Event-Graph Language for representing procedural knowledge.

A set of support tools (editors, compilers, interpreters, etc.) has been identified for each of the above languages. A basic subset of these tools, that constitute the kernel toolkit, has been designed and is being implemented. An appropriate communication framework has been adopted, according to a methodology for using the toolkit which is being established. In the next four subsections we overview the features of each of the above languages and the relevant tools, and, finally, discuss the communication framework.

5.1. The Component-Based Language

The Component-Based Language (CBL) has been designed for representing ontologic models of technical artifacts in such a way as to allow different reasoning activities about the behavior of the system (like prediction, diagnosis, control, etc.) to be automated. In particular, a reasoning mechanism is currently being designed for tasks of prediction and diagnosis.

The basic concept of the CBL is that a physical system is specified by describing its internal structure as a set of interconnected components. A model is defined for each component from which the component's behavior can be inferred, giving the behavior of the whole system. Thus the CBL provides a definition section, for defining abstract models of components, and a declaration section, for describing an actual system in terms of a set of component instances, and their interconnections.

A component type, i.e. an abstract model of a component, features a definite number of terminals of a given type. Terminals are ports through which components are to be interconnected; a terminal type is a structure whose elements are scalar variables representing measurable quantities. For instance, electrical components like resistors, capacitors, coils, etc., shall be connected through terminals representing an electron flow, and featuring voltage and current intensity as scalar elements.

The component model is provided as a set of equations constraining the terminals variables, and some component specific parameters (e.g., the resistance in a resistor, etc.).

Equations may be interpreted either in quantitative or in qualitative terms using specified domains. Primitives are available for allowing the user to introduce his own domain definitions, however some domains are predefined in the language, in particular the (quantitative) domain of real numbers, and a qualitative domain allowing a set of three values: (inc, dec, std), which represent the qualitative rate of change of a variable with respect to a reference value.

Moreover, the CBL provides constructs for specifying subparts into a component; and for defining different views about a component, which specify the component behavior respect to different physical processes (e.g. the electrical and the thermal view about a resistor); and for defining a component type as a specialization of a wider class, having a more abstract description.

The basic features of the definition section of CBL are summarized in the following example describing an hydraulic circuit element:

```
DEFINITION-MODULE hydraulic-and-thermal-components

        CONSTANTS IN-DOMAIN real
                (g = 9.8)        ; gravity acceleration
                (ro = 1.)        ; fluid density
                (c = 1.)         ; water specific heat
                (u = ..)         ; water viscosity

        END-CONSTANTS


        STRUCTURE hydraulic-characteristics
                G        ; flow-rate
                P        ; pressure
                T        ; temperature
                v        ; velocity

        END-STRUCTURE


        COMPONENT-CLASS  hydraulic-circuit-element

          TERMINALS in, out OF-TYPE hydraulic-characteristics

          PARAMETERS
                R        ; friction parameter
                h        ; heigth difference between input and
                         ; output section

          RELATIONS
              IN-DOMAINS real
                in.v = out.v
                in.G = out.G
                out.P = in.P + ro*g*h - R*in.G**2

             IN-DOMAINS qual
                in.v = out.v
                in.G = out.G
                out.P = in.P + h - R - in.G

          VIEW thermal-dissipation
                PARAMETERS
                        delta    ; temperature diminishing
                                 ; because of dispersion

                RELATIONS
                    IN-DOMAINS real, qual
                        out.T = in.T - delta

                DEFAULTS
                     IN-DOMAINS real, qual
                        out.T = in.T

        END-COMPONENT-CLASS

        .....

END DEFINITION-MODULE
```

The CBL declaration section allows the user to describe a specific physical system by declaring its components as instances of component types provided by means of the definition section; and to state the connections between component instances. Views about component behavior may also be instanced (but it is possible to switch on different views at run-time, i.e when the model is being used).

Current limitations of the CBL are mainly relevant to the language predefined operators available for quantitative and qualitative domains. Our discussion will be restricted to the predefined qualitative domain, a distinctive feature that CBL inherits from similar representation languages introduced by AI research in qualitative physics (Bobrow and Hayes, 1984). Qualitative models have two basic advantages over quantitative models: they are simpler to compute and provide results far easier to interpret. However they have some limitations (Kuipers, 1986) which are not completely evaluated in particular with respect to prediction of system's dynamic behavior. For that reason, CBL qualitative equations syntax currently allows expression of algebraic relationships only. Hence the CBL allows representation of static (equilibrium) system models, and reasoning about such models is restricted to considering displacements between equilibrium states. Based on static models, it is however possible to perform diagnosis of system malfunctions in a large variety of practical cases. It is recognized that such a variety requires different strategies for performing diagnosis. Thus we intend to provide the CBL with a number of inference mechanisms, each one suitable for a particular diagnostic strategy. Similar conclusions apply to control tasks, where, however, the deduction of system's behavior over time is mandatory, hence an extension of language syntax will be required to include dynamic operators.

5.2. A diagnostic problem solver using CBL

Currently an in-depth analysis of one class of diagnostic problems has been performed, where diagnosis consists of determining which variation of a system parameter has caused a change in the stationary state of operation of the physical system, using explicit fault models, which correlate each known system malfunction with variations of known system parameters (Gallanti, Gilardoni, Guida, and Stefanini, 1986). Explicit fault models exist, which correlate each known system malfunction with a variation of a system parameter. A suitable inference mechanism has been designed, and has been implemented within the kernel toolkit. This is composed of two subsystems:

a) a causal analyzer, performing a static analysis upon system representation, under the viewpoint of the problem to be solved (i.e., symptoms observed and fault hypotheses). Causal analysis allows to ascertain whether the proposed problem is solvable, and, if it is, to build a net representing causal dependencies between hypothesized causes and symptoms. This net drives the subsequent diagnostic process.

b) The diagnostic problem solver. This subsystem is in charge of evaluating consistency between the hypothesized fault causes and the observed symptoms. It is based upon well-known techniques for performing hypothetical reasoning upon

a set of equations: constraint propagation and truth maintenance. The relevant modules, a Constraint Propagator (CP) and an Assumption-based Truth Maintenance System (ATMS), have been specified and prototyped, following the indications of Sussman and Steele (1980), and De Kleer (1986), respectively.

We foresee now to design a more general diagnostic problem solver, for diagnosis of multiple faults, avoiding the use of explicit fault models, considering instead as a fault any behavior which is different from the expected one. The task of the diagnosis is then to find the candidates, i.e. the components responsible of the fault(s) and consistent with the observations made so far.

5.3. The rule-based representation language

Over the last decade rule-based languages have attracted much attention as one practical solution for expressing human knowledge. Most ESs, and almost all the available ES development environments, use production rules as a basic representational tool. Hence, a rule-based language is included within the toolkit.

However, available production systems differ greatly under many different respects, which can be roughly summarized as such:

a) the rule syntax (and the implied semantics), a wide range of solutions is available, ranging from pure first order logic to several viable solutions for treating uncertainty of predicates and inference;

b) in particular, predicates in the rule's premise and conclusion may refer to a data base where data have either a flat, unstructured representation (like in first order languages), or may be arranged into data structures of several kinds. Under this viewpoint, several KBS development environments among the most advanced available on the market, e.g. Inference Corporation's ART and Intellicorp's KEE, couple a rule-based language with an object-oriented language.

c) Moreover, rule interpreters (which in most cases operationally define the semantics of the language) offer a great variety of matching algorithms, conflict resolution and search strategies.

The definition of the toolkit rule-based language has been very pragmatic, as we have chosen to basically mirror the solution provided by the environment adopted for prototyping the kernel toolkit. This is one of the high level, general purpose, KBS development environments available on the market mentioned above. Again, our choice of KEE among the candidate environments that have been considered, is based on pragmatic reasons like user-friendliness and availability on a wide range of machines, has been motivated by the need of rapidly prototyping the kernel toolkit and the demonstrators during the first phase of the project.

A further analysis of the demonstrator requirements, relevant in particular to regulatory control tasks, has allowed to identify and specify an additional rule-based formalism, very

different and far more specialized than the one provided by KEE. This further formalism is oriented to treat data vagueness and adopts for this purpose the fuzzy logic approach. It will be hereinafter referred as the fuzzy rule-based language, while the general purpose production rule formalism will be named binary rule-based language. We devote the remainder of this section to a brief presentation of the task the fuzzy rule-based language is suitable for, to a discussion of the problem of treating data vagueness and the fuzzy logic approach, and to a short overview of some distinctive features of our solution.

We have noticed that many continuous control problems that defy formal analysis are rather easily controlled by human operators basing on empiric knowledge. This knowledge is often expressed in vague terms. Let's consider an example taken from the expertise of operators of the kiln in a cement manufacturing plant: "if the burning zone temperature is drastically low and oxygen level in exhaust gas is low, then reduce the kiln speed and reduce fuel". The continuous aspect of the regulation is stressed. The experts find it difficult to fix thresholds upon which the predicates mentioned in the premise of a rule like the above should be evaluated true; the rule should be checked if not continuously, at least periodically, and the regulations mentioned in the rule conclusion should be operated in a way proportional to the truth of the premise. Fuzzy logic is a qualitative approach to the representation of human expertise like the one mentioned above. Natural language terms, such as low, high, etc., are used as labels for membership of variables defined on a universe of discourses. The membership is expressed as vague or gradual in contrast to binary logic where the transition between non membership and full membership is abrupt.

Fuzzy logic is useful when decisions are a set of compromises between several contradictory demands rather than either-or decisions like in a binary rule-based language.

A fuzzy rule-based language is, like a binary rule-based language, basicly a set of premise-conclusion rules of the form:

P1 & P2 & ... & PN IMPLIES C1 & C2 & ... & CN

The premise-conclusion terms P1, P2,..., PN, C1, C2,..., CN may be considered as a conjunction of fuzzy logic terms evaluating to a membership expressing the degree of truth.

As opposed to binary rules, fuzzy rules always apply, with some degree of truth. In forward chaining the degree of truth of the conclusion depends on the degree of truth of the premise. In backward chaining the degree of truth of the premise depends of the degree of truth of the conclusion.

The fuzzy rule-based language adopted for the kernel toolkit is based on the proposal of Zadeh (1973). The language interpreter features a forward chaining mechanism, which results better suitable for regulatory control tasks. Implication is defined as the product of the membership functions of premise and conclusion, according to a technique that has proven its validity in fuzzy controllers operated by F.L.Smidth (Holmblad and Oestergard, 1982).

5.4. The Event-Graph Language

It is recognized that knowledge relevant to many of the tasks mentioned in section 2.2 is often expressed and manipulated in a procedural form. For instance, knowledge used to perform diagnosis involves description of procedures, sequences of testing and repairing actions as well as sequences of physical components to verify, connect or disconnect. It also may include description of possible evolutions of a physical system, whether these evolutions are normal or abnormal. These evolutions, which can be concurrent, need to be checked, recognized and tracked.

A similar need arises in sequential control, where it is necessary to order and to link together several actions (e.g., start-up and shut-down procedures).

In the examples above, knowledge is such that knowledge items are ordered in a structure, whether this structure is based on causality, precedence or other ordering criteria. In this case procedural knowledge is relevant to sequencing and coordination between several actions.

While it is true that these types of knowledge can be represented thanks to a rule-based system, it is also true that in most of these cases, rule-based representations are innatural and difficult to maintain, and inference mechanisms are unnecessarily overloaded because the search space is not properly partitioned.

In conclusion, if the knowledge is structured, it is always worth putting this structure explicitly in the representation. An appropriate representation formalism, in addition to providing a mean for expressing sequencing between facts and actions, should cope with the following requirements:

- Concurrency.
  Actions can be done either in series or in parallel. Likewise, sometimes a system can reach some given state either in sequence or contemporarily. Thus the language must provide primitives for expressing local non-determinism within a procedure.

- Abstraction.
  The language should enable the user to adopt a top-down approach. The description or specification of a given procedure should be general at first, then each specified action of the procedure would be further detailed as a lower-level procedure, and so on. In other words, the user should be able to give his knowledge at different levels of abstraction.

- Hierarchy.
  One important and peculiar instance of abstraction is the concept of hierarchical control. That is to say that a procedure can be expressed for looking at, checking, modifying, managing other procedures.

- Dynamic representation.
  For controlling run-time the state of a system, or the execution of a procedure, it must be possible to know step by step which state the system is into, or which action has

just been executed. Thus our representation should allow to clearly identify the state of execution of a procedure. We want to remark that most imperative languages do not cope with such an issue, or cope with it only in an indirect way.

The Event-Graph Language introduced in the kernel toolkit shares several features with CE Petri nets (Reisig, 1982), a formalism compliant to all the requirements listed above.

We disregard here the formal definition of an event-graph and only focus on its basic expressive features.

An event-graph is a directed graph with two types of nodes, namely places and events, and a marking concept. Nodes termed places are graphically represented as circles and nodes termed events are graphically represented as boxes. Directed arcs can only connect nodes of different types. Events are labeled with two expressions, specifying a condition and an action respectively.

A place can contain a mark. An event-graph can have any arbitrary number of marks. The set of all marked places defines the current marking of the event-graph. An initial marking is defined for each event-graph.

An event is enabled if all its ancestor places, called the input places, are marked. An event can fire, ie it is active, if it is enabled and its condition is true. We define as current conditions of an event-graph the set of conditions labeling the enabled events. For example, the current conditions of the event-graph shown in Figure 1 is {c1}.

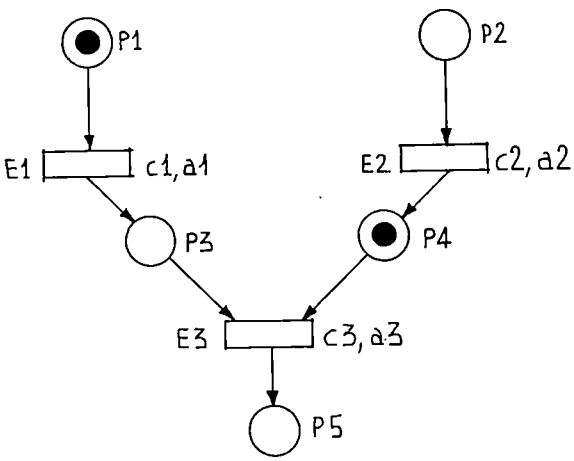


Figure 1 - An example of event-graph

An event-graph is active (respectively enabled) if at least one of its events is active (respectively enabled). Firing an event means unmarking its input places and marking all its successor places, called output places. Firing an event also causes the execution of the action specified in the event. Current marking thus evolves by means of event firing.

We stress that the concept of event-graph embodies a static and a dynamic part. The static part, i.e., the definition of places, arcs and events with conditions and actions (plus the initial marking) represents the code of a chunk of procedural knowledge.

The dynamic behaviour of an event-graph is represented by the sequence of current markings that the event-graph goes through as a consequence of successive events firing. Thus, the static part of an event-graph may be activated in different contexts and produce several images corresponding to different current markings, similar to a re-entrant procedure which can be executed several times in parallel with different parameters.

Hence, the dynamic behaviour of such procedures must be easily observed and influenced from outside. This meets the requirement about dynamic representation introduced above, which is crucial when procedures are important not only for the results they can compute, but specially for the intermediate computations they perform.

5.5. Communication within the kernel toolkit

The issue of communication within a composite environment, like the toolkit, has many different facets. Among the most important we mention:

a) identifying a communication protocol among the tools provided by the toolkit, in order to enable toolkit users to structure composite KBSs, making use of several toolkit formalisms;

b) providing the user with a control policy (and a supporting mechanism) for managing the interaction between several knowledge sources;

c) establishing a methodology for using different models (empirical and ontological) in order to perform a given diagnostic or control task;

d) defining a methodology for representing and structuring the interfaces between several software packages which constitute the elementary building blocks of the individual tools provided by the toolkit.

These different aspects are however so closely related that it is difficult to cope with each of them separately. On the other hand, it is overambitious to provide the kernel toolkit with a complete solution before than sufficient experience has been achieved with the several formalism provided by the kernel toolkit, especially coping with real size problems within the demonstrator projects. This impacts in particular on points b) and c) introduced above. Therefore it has been decided to provide the kernel toolkit with mechanisms for communication, instead of policies.

In other words, we have provided a solution to problem d) above, and ways to cope with problem a), while the most complex issues relevant to problems b) and c) will be faced in the second two-years period of the project, when the final toolkit will be specified. The point can be better understood referring to

Figure 2, illustrating the overall development process, both of the toolkit and of a composite KBS built using the toolkit:

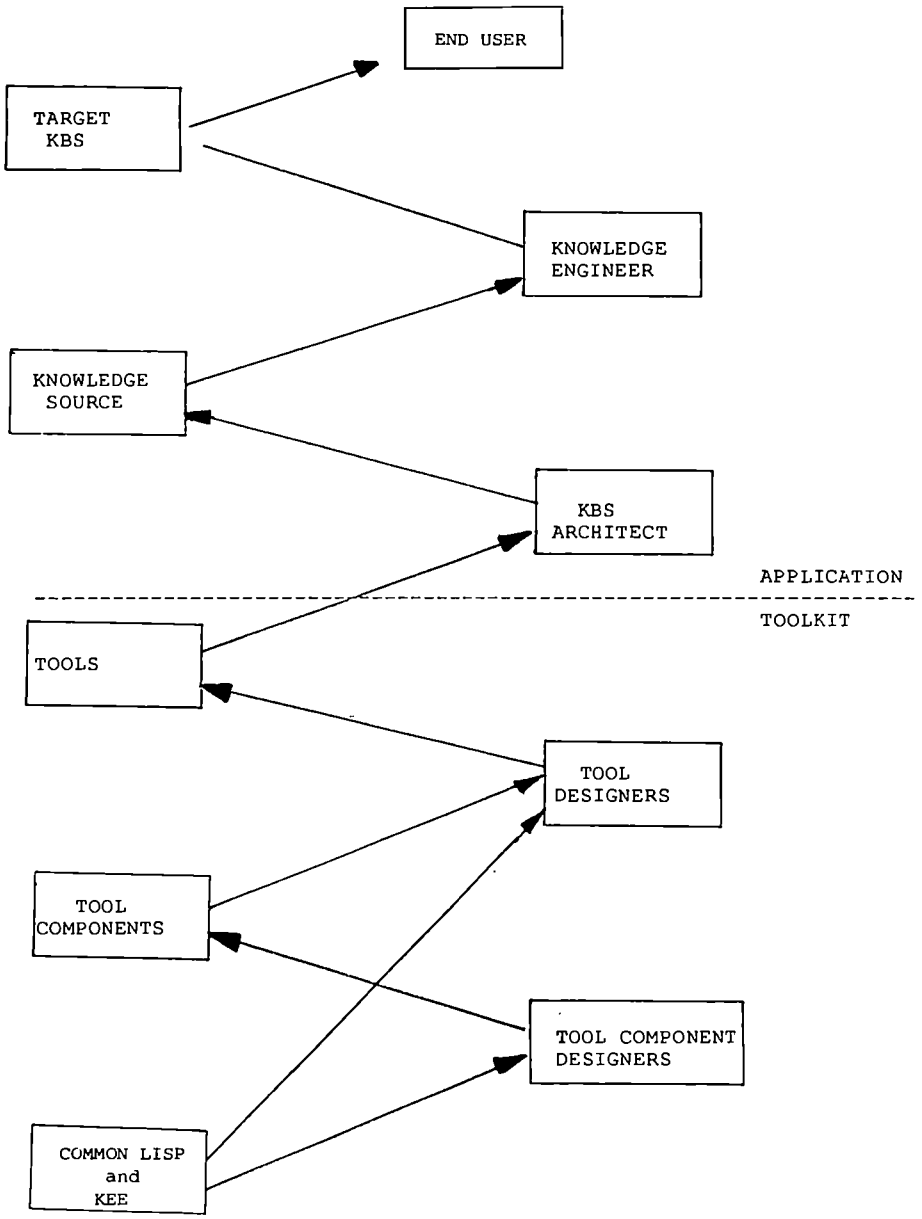


Figure 2 - Layers in the development process.

From the toolkit side, a toolkit designer integrates several tool components, each one providing a well definite function,

into ready-to-use finished tools. These are complex environments, providing full support for performing a given task using a definite toolkit formalism. The fuzzy rules interpreter, the Event Graph processor, the diagnostic problem solver using CBL are examples of such finished tools, while the causal net generator, the constraint propagator, and the assumption-based truth maintenance system integrated into the diagnostic problem solver are examples of tool components.

From the application side, we postulate a KBS architect in charge of defining the overall architecture of a KBS application as a collection of communicating knowledge sources. The knowledge base of each knowledge source will be actually provided by some individual expert or knowledge engineer, and will be represented with a definite toolkit formalism, thus making use of the capabilities provided by some finished tools.

6. OPEN ISSUES

We have already stressed in section 4.2 one of the major limitations of the component-based language currently being developed: it does not allow deduction of the system behavior over time. Many research efforts have been devoted to qualitative simulation of dynamic systems in recent years: the most comprehensive work being provided by B. Kuipers (1986). However, qualitative dynamic simulation methods currently available generate, together with the system's proper behavior, many spurious solutions that redder qualitative simulation almost ineffective in most non trivial applications. As prediction of behavior over time is crucial for almost all control applications, the issue is the subject of intensive research within P820.

On the other hand, the role of qualitative dynamic simulation in diagnosis tasks is still to be completely understood. More in general, many recent studies in qualitative modeling have been oriented towards diagnosis (e.g. De Kleer and Williams, 1986), so that it is now necessary to review these newly emerging solutions, in order to provide the component-based language with problem solvers able to pursuit diagnostic strategies different from the one which is now included in the kernel toolkit.

Another task of great importance in process applications is planning, especially for sequential control. In this sort of applications, representation of time dependency and reasoning about time relations is crucial. Among many solutions, the temporal reasoner proposed by Allen (1983) is one of the most effective and sound, being based on a classification of time relationships and a set of relevant inference rules. Research in P820 is aimed at providing a time reasoning system upon which to base a planning module, possibly exploiting the basic inference machinery already available within the kernel toolkit, i.e., a constraint propagator and an assumption-based truth maintenance system.

The above research efforts take place in parallel with the development of the kernel toolkit and of the demonstrators, in order to provide results when the design of the final toolkit will be initiated.

REFERENCES

Allen J.F. (1983). Maintaining knowledge about temporal intervals. Communications of the ACM26,(11), 832-843.

Bobrow D.G. and Hayes P.J. (Eds.) (1984). Qualitative reasoning about physical systems. Artificial Intelligence Special Volume 24, (1-3).

De Kleer J. (1986). An Assumption Based TMS. Artificial Intelligence 28, 127-162.

De Kleer J. and Williams B.C. (1986). Reasoning about multiples faults. Proceedings of the National Conference On Artificial Intelligence. Philadelphia, PA, 132-139.

Gallanti M., Gilardoni L., Guida G. and Stefanini A. (1986). Exploiting Physical and Design Knowledge in the Diagnosis of Complex Industrial Systems. Proc. 7th European Conference on Artificial Intelligence, Brighton, UK, 335-349.

Holmblad P.L. and Oestergaard J.J., (1982).Control of a Cement Kiln by Fuzzy Logic. Internal Report F.L. Smidth and Co., Copenhagen, Denmark.

Kuipers B., (1985). The Limits of Qualitative Simulation. Proc. Of the Ninth International Joint Conf. on Artificial Intelligence (IJCAI-85). William Kaufman. Los Altos, CA.

Kuipers B., (1986). Qualitative Simulation. Artificial Intelligence 29, 289-338

Reisig W., (1982). Petrinetze. Springer Verlag, Heidelberg, FRG.

Sussman G.J. and Steele G., (1980). Constraints - a Language for Expressing Almost Hierarchical Descriptions. Artificial Intelligence, 14, 1-40.

Zadeh L.A., (1973). Outline of a New Approach to the Analysis of Complex Systems and Decision Processes IEEE Transactions on Systems, Man and Cybernetics, Vol. 3, No. 1, 1973.

Project No. 1542

INDOC - INTELLIGENT DOCUMENTS PRODUCTION DEMONSTRATOR

A. Celentano (1), P. Paolini (1,2)

(1) ARG SpA, Via Fratelli Bronzetti 18
20129 Milano, Italy
(2) Politecnico di Milano, Piazza L. da Vinci 32
20133 Milano, Italy

A large number of applications need to handle documents with the following distinctive features: *complex structure*, *integration of text with 'values'* (either formatted data or images), *relevance of semantics* (mistakes are absolutely to be avoided), *range of complex rules* to be followed. Typical *application environments* where such documents are found are *legal offices*, *public administration offices* (at any level), *large organizations* (e.g. banks). Typical examples of documents are *contracts*, *loan documents*, *administrative acts* etc. Typical examples of *rules* are *law*, *administrative rules*, *tradition* (within an organization or within a group of professionals), etc.

Current automation technology has devoted a great effort to help to shape the 'appearance' of documents, while has paid little attention to the content. The traditional handling of the above type of documents shows a number of serious drawbacks: only specialists really understand the semantics of the documents; skilled people are required to fill them; few experts know why the documents were structured in a given way; changes of rules become painful; major restructuring becomes virtually impossible (it is a real life experience that large organizations loose control over the huge amount of documents, of all the possible sorts, which they routinely produce;.

INDOC, which is based on a specific industrial experience on automation of legal offices, has the following objectives:

- to allow the definition of document *generators*, which can produce, in a data driven fashion, highly sophisticated documents. This will allow less-expert personnel to produce high quality (in the sense of semantic complexity) documents

- to split the design (and production) of the *content* of a document from the design (and production) of its *appearance*. This allows: focusing on relevant issues, semantic management of documents and interchange of documents between different experts (even if they speak different languages)

- to build an *expert system*, which 'captures' the body of rules governing the documents; its purpose is to speed up *training* of new personnel and support *maintenance* with *explanations* ('why do we have this clause in this contract?') and *bindings to rules* ('if this law has changed, which documents should be reviewed and how?').

1. WHICH KIND OF DOCUMENTS

The documents processed in the INDOC environment come from juridical, legal and business applications. Examples can be considered legal contracts for sales or for incorporation of a new company, the documents of a bank concerning a loan, the administrative papers released by a Public Administration office etc. These documents exhibit a number of distinctive features which allow us to use the term *intelligent document* to refer to them:

- *semantic relevance*: documents of this kind (for transferring the property of a real estate, establishing the conditions of a mortgage loan, expressing a decision of a city council about new expenses, or establishing the conditions for exporting goods) embed statements which describe facts of social relevance, and cause important effects to be taken on the outside world. Therefore even minor mistakes must be avoided, since the effects could be very serious (e.g. a wrong word at the wrong place may cause the invalidity of a contract)

- *structural complexity*: these documents are usually highly structured. The structure is very often imposed by specific rules. Legal documents, for example, require a specific order of sentences and a formal use of words, and in many countries they require also a specific type of paper form, with a standard set of formatting rules

- *rules driven*: a large body of rules must be applied; the body of the rules is often partially formally stated and "official" and partially informally stated. The formal rules are reflected in the interpretation of laws and regulations suitable for the specific situation the documents describe. The informal rules arise from the fact that every professional or organization follows a set of "personal" rules, by experience, habits, or style, which are to be considered as binding as formal rules

- *patterns*: the documents follow predictable patterns. In a given application area, they can be subdivided in types, each type identifying the features which are common to a class of documents to cover a specific case. The patterns can be very complex, each type being possibly made of variants, which can be considered subtypes. The variants, for example, could specify the inclusion of exclusion of clauses, several versions for expressing the same concept, different number of instance for iterative parts, and so on. At the lowest level of specialization, documents differ only for specific occurrences of variable data, such as names, prices, references to other documents, and so on

- *external information*: the documents incorporate a large quantity of information (e.g. about people, estates, prices, and dates) which in general have other operational uses, beside appearing in the documents themselves. These data could be present in some Information System for purpose other than producing the document, or could be used in several (related) documents with different presentation styles. The

information, although in some sense "foreign" with respect to the documents, carry their actual semantics, and the text has the main purpose of making this semantics clear to the reader.

In the next section we will examine the process for producing documents when simple word-processing technology, or no technology at all is being used.

2. HOW DOCUMENTS ARE PRODUCED

The procedures followed for producing documents, in absence of a specific information technology, reveal similar patterns in different organizational environments, such as professional offices and banks. As an example, we briefly examine the behaviour of Notary offices, which in our experience is a highly representative case.

When a new situation requires a document, the professional fits the real case into a technical classification: the reasoning follows patterns such as (in a trivial example) "since Mr. Rossi wants to sell his house, a real estate sale type of document is needed, and the clauses for the sale are so and so...". This type fitting activity is performed at various levels of precision: first the type "real estate sale" is identified; then, for example, the subtype "with no mortgage" is selected, and so on. At this stage two behaviours are possible:

synthesis: the document is synthesized, basically from scratch, choosing the appropriate sentences which state in a clear way the concepts appropriate for the identified situation (with its possible variants and subvariants)

analogy: a model, i.e. a prototype document for the needed type is selected and modified, in order to fit the specific situation. We need to distinguish two types of changes of a prototype: semantic modifications (e.g. introduction of a clause for mortgage) and factual modifications (e.g. change of a date, or a name, or a price, etc.), that we assimilate to the activity of putting data into proper places (see below).

The synthesis is supposed to give the best result, but our investigations show that it is very seldom used (but in the schools, for training purpose). The analogy is typical in real life situations. It should be noted that the handbooks for professionals (e.g. notaries and attorneys), which collect the large part of the clauses, in essence are structured as a set of prototypes of documents. A number of features are common to both approaches:

- a document is made up of chunks which we call clauses; each clause can be made of other clause
- each clause has a content, which expresses its semantics (e.g. that Mr. Rossi is the seller, that the house is located in such a place, etc.), and a presentation, which expresses the specific words used to surround the information (e.g. a price) and the specific way to display the information (e.g. digits or letters)

- the body of rules, which (more or less consciously) the professional bears in mind, specifies the (semantics of the) clauses to be used, the way to present them, the way to arrange them (that is the logic ordering of the document)
- data must be collected and put in proper places within the document; the choice of which data to collect is (partially) determined by the type (and subtypes) of the document being produced).

The expertises needed include *understanding the real case*, then *identifying the proper prototype* along with the *semantic modifications* required to adapt it to the specific situation. Also the *identification of relevant data*, and their *correct positioning* within the document, can be a difficult task if the prototype has been significantly changed.

A further complication arise from the fact that some of these activities are performed by the professional (who is supposed to own all the needed expertise), while some other activities are performed by the secretary (who usually has clerical skills, only).

From the above situation a number of problems arise:

P1 Prototypes must be kept simple, since otherwise the secretary will not be able to handle them; therefore the variants within a prototype must be few and easy to select.

P2 Problem P1 has the consequence that a large number of prototypes must be produced often just to accommodate similar cases, which, conceptually, could be considered as variants of the same case.

P3 Problems P1 and P2 combined have the consequence that only the major cases are covered by the prototypes (something like 80%); the other cases are handled as exceptions (modification of a prototype), although conceptually they could have been anticipated.

P4 Every time a modification of a prototype is needed, the activity of the office is significantly slowed down, since a number of new directives must be given by the professional and executed by the secretary; some of these directives can be very detailed such as "write these words right here, and fill in these data there".

P5 Every modification of a prototype is not only inefficient but also a possible sources of mistakes. One of the reason is that the professional must bear in mind several facets of the modification at the same time:

- the concept behind the modification (e.g. we need the description of the mortgage)
- the words to be used to express the concepts
- the identification of the new data which must be collected
- the positioning of the data within the new clause
- the positioning of the new clause(s) within the overall document
- the evaluation of the impact of the new clauses in the global architecture of the document.

The last item of the above list is probably the main reason why modifications typically add something and seldom delete anything.

P6 The deep "rules reasoning" behind the structure of the prototype could be not completely known to the professional, mainly if the prototype comes from handbooks or colleagues, or it has been drawn up from a document; the rules reasoning behind the modifications is often more under his control.

P7 The professional seldom records his reasoning behind the modifications, therefore with time he may loose control over them.

P8 The combination of P6 and P7 makes very difficult to perform operations such us "cleaning the house",i.e. revising the set of prototypes and modifications created in a number of years and rationalize them.

P9 The combination of P6, P7 and P8 makes very difficult to timely and precisely react to a major modification of the rules (e.g. new law), evaluating its impact on the prototypes and on the modification typically used; this is another reason why legal documents grow with time and never shrink: people feel safer in adding something than in deleting.

3. DOCUMENT PRODUCTION

The analysis exposed in the previous section leads us to a number of conclusions:

- the notion of a prototype for a class of documents is largely present both in small organizations like professional offices and in large organizations like banks and companies. The production of a document is not a creative activity but it reflects standards and constraints which are defined once in the prototypes and consistently used

- the dependence of a prototype on the originating rules is not always clearly stated, nor it is always preserved along the prototype's modifications

- the intermixing of semantics of text and its presentation is a source of confusion

- if an efficient way could be found to incorporate the knowledge on how a document should be produced, once the case is identified and the data are available, then the document production could become a routine task. In essence is the goal is to substitute a "passive" prototype with a more "active" schema

- a link between the internal rules applied in a schema and the "external" rules (laws etc.), is needed in order to help the huge problem of maintenance and revision of large classes of documents.

Therefore we base our approach upon three leading *ideas* (notions): the idea of separating the *semantics* of a document from its *presentation* (appearance), the idea of capturing in a *schema* all

the rules for generating a class (type) of documents, the idea of using an *expert system* to support the *handling* and the *maintenance* of a large set of schemas.

In the following of this section, the first two ideas are analysed, while the third is dealt with in the following section.

The semantics idea

The description of a document could be done at several levels:

- description of the *physical appearance* of the document elements with respect to their final layout (e.g. line numbering and sizing, footers, headers, and so on); this description is captured by the layout structure in the standard ODA terminology

- description of the *structure* of document elements (e.g. introduction, chapters, paragraphs, and so on); this description is captured by the logical structure in the standard ODA terminology

- description of the *semantics* of the document, that is to say the meaning of information carried by the document to the reader; this description is missing in the ODA classification.

In order to proper model documents, in INDOC, we need to distinguish between a *conceptual model* and a *presentation model*. A conceptual model is able to convey all the meaning of the document itself without concerning about (in effect hiding) all its logical and layout aspects. The presentation model, on the other hand, captures the way the document looks alike, with little or no concern to the concepts it expresses.

Since the conceptual model captures the meaning of a document, it can be used in a number of ways:

- as a way to formulate the information requirements of a document: since I want to express these concepts, which information need I to collect?
- as a vehicle to quickly understand the semantics of the document: what is the document about?
- as a bridge between the actual documents and the application rules: why this concept is expressed?
- as an intermediate step in document production: by separating semantic aspects from logical structure and style it can support different logical structures and presentations of the final document and can also be a suitable vehicle for exchanges among different organizations or professional offices.

The last use is the prevalent one in INDOC, and will be further described.

The information belonging to the conceptual model of a document can be intuitively split in two parts:

- simple factual information; for instance, data about entities (people, properties, and so on) and their attributes (names, locations, prices, and so on)

- relevant semantic information; for example, stating roles of persons, that is establishing who is the vendor, who is the buyer, which form of payment a price describes, and so on.

The distinction between the two types of information is purely pragmatical: in general there is no unique way to define a border between simple facts and complex concepts. However, such a border can be traced on the basis of the following statements:

- there is a general agreement, though not formalized, among the application experts on the assignment of information to each type
- the application experts find easy to make the above assignments
- the modelling activity can be performed more efficiently if a clear distinction is made between the two types of information

Relevant semantic information can be described by *assertions* about entities and values: for example the fact that "*in document D person P1 conveys the property of the real estate E1 to person P2 at price $*" could be a set of assertions formally stated as

```
OBJECT(D,E1)
VENDOR(D,P1)
 BUYER(D,P2)
PRICE(D,E1,$)
```

This the essence of the representation chosen in INDOC, for the conceptual model of a document. Several real-life examples have been worked out, using slightly different notations. One of the encouraging result of this effort has been the reactions of the "application expert": after a short initial diffidence, they considered the conceptual models as semantically equivalent to the original documents, with the advantage of being shorter (1 to 5 is the approximate ratio), more compact and "easier" to read.

The schema idea

What we have described in the previous section is the framework for modelling a single document; however models are useful if they can uniformly be applied to several documents (document instances). This introduces the notion of *schema*: a schema defines the structure and the content of a set of documents. The composition of a schema combines pieces of text and data (which are the tools for clarifying the concepts carried by the document) in order to produce the instance of a document. A schema should not be intended as "a text with holes, to plug in data"; it is rather similar to a program which, according to the value of data, selects pieces of text to be produced (one or more times), plugging in data properly formatted.

In INDOC we consider two possible types of schemas: *conceptual schemas* and *presentation schemas*. A conceptual schema describes the transformation operated upon a set of basic information, that we call *Structured Universe of Discourse*, to generate a conceptual model of a document. A presentation schema describes the transformation operated upon a conceptual model of a document, to generate the presentation model, which can be thought as a "real" document, in the usual meaning.

It should be noted that several different presentation schemas could be applied to the same conceptual model, generating several different final documents (with the same underlying semantics). Interesting application seem to arise: the same document generated in different languages, the same document generated in full version and in synthetic version, etc.

Operationally speaking, the document production in INDOC is conceived in the following way:

- *define a schema for a class of documents*: this activity is performed only once, at configuration time for a specific application

- *provide data for the specific case*: this activity can be performed in different ways according to the data management solutions adopted. Data bases can exist for other purposes, and they could have been populated before (Think for example of the data concerning the customers of a bank, which exist independently from documents where they are referred). In this case the process which generate the document must be interfaced with the preexisting Data Base. In other cases data are not necessarily supposed to be collected elsewhere, so the generator itself could ask for them during the generation process

- *execute the composition of a document*: this is obtained applying a schema to a set of data.

The above steps apply both for conceptual schemas and for presentation schemas.

4. THE UNDERLYING KNOWLEDGE

Behind the structure, the content and the physical appearance of the documents it can be found a set of motivations which rely both on official statements, expressed for instance by Laws or regulations internal to a given organization, and on the rules of behaviour that are followed in the professional practice. Official statements or rules of behaviour may describe which sentences or data must be expressed in a document to get the required effects, how they must be organized inside the document, and which must be their physical appearance.

In most cases these motivations are kept implicit in the documents (except few cases in which some of the legal motivations must be explicitly reflected in the document, for instance through the reference to very specific articles of law which must be applied).

Whether implicit or explicit these motivations represent the knowledge needed to understand the "deep" semantic carried on by the documents:" why a given document has a given content, structure, or concrete lay-out? are there any more practical or legal effects of the document besides those explicitly stated in the document? what happen if a given part of a document is modified, or a mistake occurs in it? when an official regulation changes, which types of documents are involved with it and which parts of the documents must be eventually modified?".

This knowledge can be thought at several levels of abstraction. At the highest level (that we will call *deep knowledge* or *theoretical knowledge*) there are the rules that uses concepts and statements found in laws, acts of juridical or administrative institutions, regulations, etc. They are related, at several degree of abstraction and generality, to the general concepts involved in a document (for instance, the laws of the civil code which state the principles about the exchange of property; the regulation of a bank about its legal responsibility in a mortgage loan contract). They are expressed in an official and public fashion and, in principle, they can be formalized in a rigorous way. They do not prescribe any operative behaviour, even if they should motivate the operative rules or guarantee their legality.

The lower levels of knowledge (that we will call *shallow knowledge* or *operative knowledge*) are constituted by the operative rules that express how a document must be built up. For instance, in an act describing a mortgage loan contract, a rule of this kind specifies that, if the people involved in the contract agree about the absence of witnesses, this agreement must be explicitly described in the act. For the same document, there is another rule stating that for some particular banks there are particular addresses which must be used as reference addresses for mortgage loan acts.

Typically (at least in the operative environments we have analysed so far), operative rules are not expressed in an official fashion. They are rules of thumb that constitute the expertise of the professional or rules of behaviour adopted by a given organization or group of professionals. They can therefore be different in different user environments.

In some cases they do have neither any explicit nor implicit reference to the official statements of regulations or laws. they may be motivated by reasons due to traditions (thereby related to no longer existing laws), to practical convenience, to private (sometimes very reserved) policy rules.

In these cases the motivations of the operative rules are not theoretical but can be expressed by comments to the rules or by more general operative rules; this is the reason why the representation of theoretical knowledge alone is not sufficient to derive, also in principle, the operative behaviours and therefore the rules governing the document production activity.

In some other cases it is possible to discover explicit or implicit connections between shallow and deep knowledge. This requires to understand the causal or semantic relationships which links the informations at the two levels, and to uncover out knowledge at intermediate levels of abstraction.
This maieutic process is not in general a step-by-step logical procedure. The mechanisms of reasoning are very difficult to be defined, as well as the heuristic criteria which guide the experts in the interpretations of regulations and laws.

The solution adopted in INDOC is the creation of a knowledge base, in which deep and shallow knowledge are formally described, organized at several level of abstraction, and in which with the semantic links among their contents are explicitly represented.

This knowledge base, supported with skill strategies for navigating inside the network of concepts and rules, can be used as an Expert Dictionary for the following applications:

- explanation to the customer for a deeper understanding of the semantics of a document
- consultancy to the customer on alternative types of documents satisfying its requirements
- consultancy to the document productor in the identification of the generator more suitable in a given situation
- training of people who must understand why a schema of a document has a certain structure and contents (young professionals, administrative assistants of the professional etc.)
- assessment of the validity of a document schema in a given situation
- consultancy in the construction of new schemas (once the goals and the rules to be obeyed are established)
- maintenance of set of schemas to deal with changes of the rules; this would imply to be able to answer questions such as "Since R has become R', which schema may be affected? where? how? why?"

5. APPLICABILITY AND ADVANTAGES OF INDOC

A few observations can outline the (possible) advantages of the INDOC approach:

- the expertise of the people is mainly used at configuration time, to define the schemas (both at conceptual and at presentation level)

- the documents produced are highly personalized (according with the data of the data bases and with the temporary information) and yet they are correct, in the sense that they are an instance of the same schema

- the operative production of documents essentially consists of data collection (e.g. loading the data bases or providing temporary information), which can be performed by low level clerks, still producing high quality documents. Think for example of the (real-life) situation where a clerk provides some raw data (about people,estates,terms of payment) and, invoking the composition, using the proper schema, is able to produce a complex real estate sale contract, which is legally correct (if the originating schema is correct, of course)

- several version, derived from the same semantics (conceptual model) can also easily be generated.

The clear disadvantage of this approach stems from the fact that the production of a schema is a major task. All the possible situations must be anticipated and the proper choices must be made. Therefore this approach is convenient in the cases where one (or more) of the following cases occurs:

- the same schema must be used a large number of times
- the documents to be produced are complex
- the amount of data to be incorporated in the documents is large

- the semantics of the documents is relevant, in particular in the sense that a mistake can have relevant consequences.

The above difficulties lead us to the conclusion that the Expert Dictionary would be a valuable tool:

- to keep track of the rules underlying the documents
- to train people (at any level)
- to help with the problem of revision due to a change of rules
- to help to keep the growth (of the set of documents) consistent with the preexisting situation
- to help with the problem of revision due to major organizational changes.

As concrete application environments, we have analysed so far Banks (for all sort of documents), large corporations (for the documents produced by the legal office), offices of professional dealing with legal documents (notaries). We guess that similar needs could be found elsewhere.

ACKNOWLEDGEMENT

This paper is a short synthesis of the work being carried on in the project (started on January 87). We wish to acknowledge the work of people at INESC (Portugal), Mnemonica (Greece), ARG (Italy) (D. Contardi, F. Liguori, A. Pellegrini), and Politecnico di Milano (F. Barbic, S. Danieluzzi, F. Garzotto, S. Mainetti).
We also wish to tank the "experts": Dr. A. Gallizia and Dr. P. Lebano, Notaries, the National Council of Notaries and the people at CARIPLO (the sixth largest bank of Italy), for their patience and their professional support.
A special acknowledgement is due to the supervision and the help of G. Bracchi (Politecnico di Milano), R. Kowalsky and M. Sergot (Imperial College of London), J. Myllopoulos (University of Toronto) and D. Tsikritsis (University of Geneve).

BIBLIOGRAPHY

INDOC - Deliverable T1/D1 - T2.1/D1: Overall Approach Review, Application Definition and Functional Specification, Jun. 1987

INDOC - Deliverable T1/D1 - T2.1/D1: Annex 1a, A Schema for Deeds of Sale and Gift, Jun. 1987

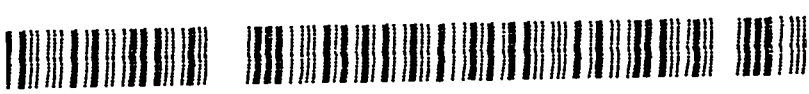
INDOC - Deliverable T1/D1 - T2.1/D1: Annex 1b, A Frame Representation of Deeds of Sale and Gift, Jun. 1987

INDOC - Deliverable T1/D1 - T2.1/D1: Annex 1c, A Schema for Deeds of Loan Receipt, Jun. 1987

INDOC - Deliverable T1/D1 - T2.1/D1: Annex 2a, Survey on Knowledge Representation, Jun. 1987

INDOC - Deliverable T1/D1 - T2.1/D1: Annex 2b, Survey on Conceptual Modelling, Jun. 1987

INDOC - Deliverable T1/D1 - T2.1/D1: Annex 3, Laws as Specifications of Objects, Jun. 1987



ISBN Part 1: 0 444 70331 4
ISBN Part 2: 0 444 70332 2
ISBN Set : 0 444 70333 0