

ICL
TECHNICAL
JOURNAL

Volume 6 Issue 3 May 1989

Published by
INTERNATIONAL COMPUTERS LIMITED
at
OXFORD UNIVERSITY PRESS

Editor

J. Howlett

ICL House, Putney, London SW15 1SW, UK

Editorial Board

J. Howlett (Editor)

H.M. Cropper (F International)

D.W. Davies, FRS

G.E. Felton

M.D. Godfrey

(Imperial College, London University)

C.H.L. Goodman

(STC Technology Ltd and King's College, London)

F.F. Land

(London Business School)

K.H. Macdonald

M.R. Miller

(British Telecom Research Laboratories)

J.M.M. Pinkerton

E.C.P. Portman

B.C. Warboys (University of Manchester)

All correspondence and papers to be considered for publication should be addressed to the Editor.

The views expressed in the papers are those of the authors and do not necessarily represent ICL policy.

1989 subscription rates: annual subscription £35 UK, £44 rest of world, US \$88 N. America; single issues £17 UK, £22 rest of world, US \$38 N. America. Orders with remittances should be sent to the Journals Subscriptions Department, Oxford University Press, Walton Street, Oxford OX2 6DP, UK.

This publication is copyright under the Berne Convention and the International Copyright Convention. All rights reserved. Apart from any copying under the UK Copyright Act 1956, part 1, section 7, whereby a single copy of an article may be supplied, under certain conditions, for the purposes of research or private study, by a library of a class prescribed by the UK Board of Trade Regulations (Statutory Instruments 1957, No. 868), no part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means without the prior permission of the copyright owners. Permission is, however, not required to copy abstracts of papers or articles on condition that a full reference to the source is shown. Multiple copying of the contents of the publication without permission is always illegal.

© 1989 International Computers Limited

Contents

Foreword <i>John Dickson</i>	407
Tools, Methods and Theories: a personal view of progress towards Systems Engineering <i>David E. Talbot</i>	409
Systems Integration <i>Roger Lucas</i>	415
An architectural framework for systems <i>P. Henderson and B. Warboys</i>	435
Twenty Years with Support Environments <i>Brian Warboys and Philip Veasey</i>	447
An Introduction to the IPSE 2.5 Project <i>R.A. Snowdon</i>	467
The case for Case <i>Fred Russell</i>	479
The UK Inland Revenue operational systems <i>E. Wilson</i>	496
La Solution ICL chez Carrefour a Orleans <i>Y. Pisigot</i>	500
A Formally Specified In-Store System for the Retail Sector <i>Val Jones</i>	511
... towards a Geographic Information System <i>J.M.P. Quinn</i>	542

“Ingres Physical Design Adviser”: a prototype system for advising on the physical design of an Ingres relational database <i>Michael Gunner</i>	557
KANT — a Knowledge Analysis Tool <i>Graham E. Storrs and Chris. P. Burton</i>	572
Pure logic language <i>Edward Babb</i>	585
The ‘Design to Product’ Alvey Demonstrator <i>Dr. L.D. Burrow</i>	598
Notes on Authors	617

Foreword

It is now clear that Information and Information Systems are becoming the life-blood of organisations.

The scale and impact of this is demonstrated in this issue, on the one hand by the experience at the Inland Revenue in the UK, where one of the largest databases in Europe is now regarded as a mere component in an organisation-wide system, and on the other by what is happening at the most recently-opened CARREFOUR hypermarket in France, where the closest integration of business needs and information systems is required for instant and local information on customer needs, stock optimisation and responsive price and margin management.

These two examples are not isolated instances but are increasingly representative of the demands of our marketplace. This marketplace is one in which customers need to manage and specify such systems as a part of their overall business process, and in which ever-increasing standards of quality and reliability are demanded.

It is appropriate therefore that this issue concentrates on system architectures and integration. Clear demonstration is given of:

- the scale and complexity of systems currently in use
- the availability of formal methods for aiding the correct specification and implementation of these systems

and

- the emergence of concepts and formalisms to support the design and management of the complex integrated socio-technical systems that will be required in the 1990s.

In addition, this issue vividly illustrates the impressive range of skills and experience available from ICL and its collaborators and the way in which advanced system theories and techniques can be rapidly applied to give real business advantage.

John Dickson
Managing Director, ICL Product Operations

Tools, Methods and Theories: a personal view of progress towards Systems Engineering*

David E. Talbot

ICL Marketing and Business Strategy, Bracknell, Berkshire

Introduction

I am writing this paper from the standpoint of an Industrial consumer of Tools, Methods and Theories. As such, one needs to be clear that the overall goal of real interest is concerned not just with software engineering, or human factors, or the inclusion of “knowledge” in systems or devices but with the predictable construction of systems which usually contain all or at least very many of these constituent elements. It is, of course, a system that in the end a customer buys, not just its software or hardware or a knowledge base or the interfaces that it presents to its users. The overall goal of work in this field must therefore be Systems Engineering.

The original Alvey Software Engineering Strategy certainly recognised this requirement from the outset. However, it also appreciated that we had a considerably less well founded position in Systems Engineering than in Software Engineering. In consequence, over its limited course of 5 years the emphasis of the programme was placed on developing and consolidating the Methods, Theories and Tools required to turn the business of producing software into a more predictable enterprise and one much less dependent on “wizards” and similar exotic creatures. Furthermore, Software Engineering was seen as a promising base from which a more general approach to building systems could be realised.

I aim, therefore, to review:

- how far Software Engineering has developed in terms of meeting its original more limited aims and how this contributes to the Systems Engineering goal.
- how far associated disciplines concerned with hardware, with issues of the human interface and with knowledge based systems have moved to encompass also the “system” view.

*This paper is based on an address given at the 1988 Alvey Conference; permission to publish it here is gratefully acknowledged.

- and above all, how far these notions are really entering the industrial "soul" and having clear and measurable effects on providing customers with systems that meet their needs – that is "Quality" systems.

The Systems Company

Let me start by proposing a view of "The Systems Company". It is one that owes much to the thinking of Peter Cropper, Brian Warboys and others in ICL who have been worrying professionally about this subject for some years.

A rather simple model would contain a number of quite straightforward notions:

- first, a view of requirements, and one, moreover, that takes the view that the elicitation of requirements is a process that requires more than just sound systematic methods and good tools, but also some understanding and sympathy with the "market" being investigated.
- next, the recurring dream (nightmare?) of "components" and the notion of component re-use as still the major way in which both the quality of the finished system can be better assured and the productivity of the engineers can be raised.
- last, the notion of dealing with components and processes over which the engineer has only limited control. It raises the issue of how design and engineering integrity can be established in an environment where the use of imported product components, and hence by implication their production processes, needs to be considered the natural way of doing business rather than the exception.

Perhaps, however, the more interesting features of such a model are concerned with points not recorded explicitly. These are:

- that the model itself is recursive and is true for successive points on an integration chain where one man's system must be considered simply as another man's component. Thus at one end of the integration chain it may be concerned with etching silicon and at the other with, say, the production of a major C³ System.
- that scale is a constant feature and that any method, theory or tool needs to accommodate itself completely to systems enterprises that are generally interesting only at levels measured in terms of tens of man years and involving the work of teams rather than individuals.
- that since successive integration is an ever present reality the chances of needing to deal effectively with complexity are high.

Against this background, let me now review developments so far.

Software engineering

Most software engineers would, I believe, have little difficulty in relating to the model described. However, most would also have some difficulty in

saying how current developments in methods, theories and tools have been of practical assistance to them in dealing with a number of the issues raised.

In the area of design the notions of "design in the large(r)" are well established. Much good work has certainly been achieved in the development of theories and methods concerned with helping deal more effectively with decomposition, improved modularity and binding, approaches to designing—in desirable properties such as reliability and such like. Good work has also taken place in the development of notations that help express design more precisely and reason about the correctness of design. These points are, I believe, of particular importance to software since software creation is, above all, concerned with the process of design.

Overall good progress has been made in this area and has been accompanied by the development of some tooling that extends just past the prototype standard. I have little doubt that over the next few years tooling will improve to match up to real industrial expectation, including a considerable increase in the quality and effectiveness of "automatic" programming. However, in pursuit of the Systems Engineering goal the need is to constantly remind ourselves that real interest lies in matters concerned with design "in the large" rather than "in the small".

Some progress can certainly be claimed in the area of requirements elicitation and specification. However, it is clear that this will continue to remain the point in the process where the most expensive errors can and will occur. There is an increasing recognition that work in IKBS and Human Factors has much to offer, but integrating these possibilities has been slow; I believe that we must continue to limit our expectation of what improved techniques and tools can deliver. This will always be an area where an understanding of the market need being investigated and the associated insights that this brings will continue to be more powerful than simply the exercise of systematic techniques supported by clever tools.

Within the model just described, the clearest single problem area is concerned with the notion of components and their re-use. Our hardware colleagues have this well in hand but it is a matter of record that the "softer" side has failed to make anything like similar progress. It is true that notions of "design blueprints", general application packages, system kernels etc. have come or are becoming a well founded feature of the software scene. However, it remains the case that the systematic and natural use of components at whatever level continues to be a weak spot. Moreover, there is increasing recognition that this is less of a technical problem than a "management" and "attitudes" issue. Evidence to date suggests that solutions to this problem will need to satisfy many legal, commercial, management process and cultural points.

The final issue that I should like to touch upon here concerns the integration of methods and tools not just in the context of a single stage in the

integration chain that makes up a delivered system but across all stages of the chain. I would claim that the main integrating force should be the model of the engineering "process". My own observation here is that while we have good useful models for the engineering process at the various stages of the systems building chain, we lack good models of the overall process. Moreover, enforcing "process" in software engineering continues to be a real difficulty, relying as it does now on a mixture of brute force (exercised via project management) and dusty procedure and standards manuals.

If good design through improved techniques and tools supports the business of handling intrinsic product complexity then I would equally claim that the successful management of scale, as well as the management of project complexity, is concerned with improved theories, methods and tools that support the engineering process. "Hacking", I would contend, continues to be the norm rather than the exception in most projects.

Other contributors

That I lump these together is simply an expression of my much more limited awareness of the fields covered rather than any dismissive suggestion. I will try to make just a few observations on the very clear and essential contributions made to Systems Engineering by colleagues concerned with hardware, human factors and IKBS.

If we start with our hardware colleagues, then again I believe it is true to say that they would relate strongly to the model and its concerns over scale and its management. Above all they have made most progress with issues concerned with component re-use. However, components still tend to be "in the small" rather than "in the large". Nevertheless, the notion of components and their repeated exploitation is a natural one to the hardware engineer and he has a supporting infrastructure – legal, commercial, management and technical procedures – and above all positive attitudes.

In the area of design I would contend that the most impressive work continues to be concerned with issues of design in "the (relatively) small". Consideration of how best to handle design of large complexity moves, as predicted some time ago, towards convergence with many of the ideas developed by software engineering. Further, with designs that exploit the smallest achievable geometries, it is clear that traditional simulation methods are becoming stretched beyond the possibility of reasonable containment and hence increasing interest is being taken in the proof-based methods of Software Engineering.

Increasingly, we shall also need to develop good decision support techniques that enable guidance to be given on what functions to realise in silicon and which in software.

In the area of the management of scale and the successive stages of

integration through the establishment of well founded process, the hardware engineer is in much the same boat as his software colleague. Both have a need to improve the techniques for expressing and enforcing, in a flexible manner, orderly progress at specific stages of the integration chain; and both have a need to understand how to develop a “meta process” for the overall chain of activity.

IKBS techniques and paradigms still represent largely unfulfilled promise in the matter of improving the techniques that might be applied to design, process management and requirements elicitation and specification. Whilst there is clearly an increasing understanding on the part of both the hardware and software engineering communities that they need to pay considerable attention to the possibilities available, too many IKBS practitioners still give unsatisfactory levels of attention to the industrial requirements of predictability, maintainability and the other general attributes of a well engineered system or component.

Human Interface teams are now beginning to make real contributions to both the values of the system product and the system development process. However, from an industrial viewpoint, I would express some concern at the slowness with which this has been integrated into the standard working environment of the systems engineer. It is encouraging to see the problem tackled in organisational form in the new DTI Information Engineering Directorate. However, the techniques and tools developing will only be of help if securely integrated into the overall systems development process.

Penetration into industrial practice

We may feel encouraged therefore that moderate progress has been made in a number of relevant technical areas, but I believe we must have a lesser sense of satisfaction with the rate that these new techniques find their way into the fabric of routine industrial use.

Current surveys (e.g. NCC) continue to suggest that although take-up of improved methods is making progress, progress continues to be slow. This is my own observation. Generalisations are made about the reluctance of senior management to invest; that there is a reluctance to make any more until the use of these methods and associated investment is fully proven with all the usual “chicken and egg” effect. I believe that such generalisations are too simplistic. The USA Department of Defense commissioned a study some years ago, as part of the Ada introduction programme, on examples of the speed at which new ideas in the IT field came to be adopted in wide use. My recollection is that the time from the clear definition of the idea – not conception – to a clear position of commercial take-up – not universal adoption – was of the order of 18 years and getting longer.

In short, the implications were that we need to look at the general models of innovation that are now being explored by Business Schools and commercial

organisations having a clear interest in this subject. We in ICL count ourselves amongst such groups. Such models identify a series of subtle and complex forces at work. Any progress towards improving the take-up cycle needs to understand these forces – to reduce the negative and re-inforce the positive.

Suggestions for further study

I conclude this review with an indication of the areas to which I think most attention should be directed in future.

My first point is that I believe that much thought still needs to be given to the proper integration of the Software Engineering, Human Interface and IKBS interests. Inevitably, organisational divisions of disciplines will continue to exist: for example, systems will still continue to require hardware and architecture; and this will impede progress to improved Systems Engineering unless good mechanisms are installed to compensate specifically for what, in the end, are relatively arbitrary divisions.

My second point takes me back to the issue of the Systems Engineering process. We have made some progress towards the better understanding, modelling and control of some of the constituent processes – software life cycle, hardware development, requirements elicitation etc. – but we still have little understanding of how we might best integrate these to deliver, in a more predictable manner, well engineered system components. Further, we have even less understanding of the meta- processes that are required to deliver large scale systems involving many stages across the integration chain. I would suggest that by paying some attention to this issue we might, in addition to making progress on solving a big problem, make progress on the organisational points I have noted.

My third and final suggestion centres around perhaps the most significant issue: how to accelerate take-up of improved theories, methods and tools. Whilst it is clear that we must continue to worry about developing better metrics, gaining widespread agreement to these and undertaking trials and demonstrations to prove effectiveness and increase awareness, it is, to my mind, of vital importance that we try to understand better the relationships of this specific problem with the more general problems encountered in the innovation process. It must be clear, however, that such exploration will take us well outside our own familiar surroundings.

Systems Integration

Roger Lucas

ICL Systems Integration Strategy Unit, Product Operations, Bracknell

Abstract

This paper looks at the nature of, and market and industry background to, the emerging demand for Systems Integration. It discusses the evolving basis for competition within the industry, and the influence upon this of Open Standards. Finally, it addresses the question of the attributes of a successful Systems Integrator.

1 Introduction

Is Systems Integration capability an imperative for IT vendor survival in the 1990s and beyond; or is it simply a passing fancy of the industry, a technological hula hoop to be discarded when something else more intellectually attractive comes along?

Although there is no universal accepted definition of Systems Integration, it is perceived by customers as a key area of interest which is increasing in importance, and by the IT industry as an attractive business opportunity.

Thus it is currently fashionable for companies in the IT industry to declare themselves to be Systems Integrators.

Competition is emerging from three main sources:

- Established IT and computer vendors
- Management Consulting houses
- Relatively newly established specialist Systems Integration companies.

This paper takes an overall look at the subject of Systems Integration in the historical context of the evolving Information Technology industry. It discusses some of the issues, and suggests some of the implications for the future of vendors within the industry. It concludes that Systems Integration, in whatever guise, is here to stay because it represents yet another evolutionary step along the road of meeting the requirements of businesses, balancing the capability of the IT industry to deliver against the IT demands of its customers.

As always in this journal, the views expressed are those of the author and they do not necessarily represent official ICL policy.

2 What is Systems Integration?

The dictionary defines integration as “the making up of a whole by adding together or combining the separate parts or elements”.

In the IT context, this “whole” may be anything from a hardware box or subassembly through to a complete, managed, distributed, networked system which meets the total IT needs of a business enterprise. So the computer industry has been in the integration business from its inception; it had to be in order to deliver anything at all to its customers.

The factor which has changed with the passage of time is the deliverable itself and the level of responsibility undertaken directly or implicitly by the vendor. Progressively, over the years, IT vendors have been required to take on and manage increasingly high levels of risk on behalf of clients. In the early days it was generally the norm simply to deliver hardware and rudimentary software, to provide maintenance and support but then leave the client to sort out the rest. Although there were some high profile “bespoke” projects undertaken by vendors on behalf of their clients, these tended to be approached in a spirit of “shared exploration of the frontiers of technology”.

The evolution from those early days through into the future is discussed later, but vendors have constantly sought to deliver the highest possible “value” to their clients. This has been constrained by the levels of technology, procedure, technique and inspiration available at the time. Thus the industry can be seen to have been “riding a wave” of maximum practicably deliverable value, which has taken it, over time, into areas of increasing complexity, difficulty and risk.

In Systems Integration we have reached the point where, increasingly, the client wishes to place a contract with a single “prime contractor” for the delivery of a total business system for his enterprise. This business system will typically be required to accommodate existing systems, together with new and specially developed elements, from a variety of suppliers and combine these into a fully integrated IT infrastructure to support the business. Thus the prime contractor has sole responsibility for negotiating the price with the client and managing any subcontractors; he will undertake full responsibility for meeting the client’s timescales and functional, quality and business requirements. The normal attributes for such a project will be high risk, innovative, complex and an inability to be supplied “off the shelf” or from a single vendor.

Thus a Systems Integration contract can be defined as “a contract where a prime contractor takes total responsibility for a multi-vendor, multi-contractor IT project which delivers a client-specific solution to harmonise fully with the client’s particular business and IT environment”. The skills and capabilities needed to fulfil this role successfully are discussed later.

So, to summarise, here we have high complexity and high risk for the systems

integrator who is delivering very high value to his client's business whilst being subject to the fierce market forces of competitive tendering. Clearly this is not the type of business to undertake lightly.

However, this is just the sort of business which the market leaders are already demanding, and an area into which most major IT users will migrate in the future, albeit some much more slowly than others. Meanwhile, the formal Systems Integration contract can be viewed as the high profile, de-luxe business solutions service which must be offered by all significant vendors in the IT market. They must be able to do this, that is to be Systems Integrators, because ultimately the only alternatives will be to become either a high-volume commodity "box" supplier or a specialist niche subcontractor.

The Systems Integrator needs to be able to deliver whatever level and type of IT solution his customers require. In the short and medium term, the bulk of his business will be outside the scope of formal Systems Integration contracts. Nevertheless he will increasingly need to call upon his Systems Integration skills in order to meet his customers' more routine requirements, and without the impetus of these contracts his ability to compete at other levels will steadily decline.

3 Market background

In the past, customers have bought freestanding mainframe and mini-computer systems from companies such as ICL. The task of generating local applications and integrating these into the customer's business and computing environment has generally been undertaken by the customer himself.

Most medium/large organisations now have a multiplicity of computer systems which, together, drive the business. But these systems need to intercommunicate in order to give the organisation the maximum useful information from its dispersed data sources. Intercommunication with applications systems is necessary at system, file and terminal level; there is also a need for access to common services such as electronic mail.

In order to meet these demands, many organisations have installed corporate data networks to allow cross-connection between all relevant systems/terminals within a unified framework. Most organisations now accept that they will ultimately have to follow this path. This is a major integration task which few have yet satisfactorily completed and which most accept is beyond their experience or skills to achieve in-house.

These data networks are rapidly converging with the established corporate voice networks, and the two will logically and physically merge together into corporate information networks once suitable user terminals and networking systems are readily available.

With the advent of networked applications systems, the distinction between

applications can no longer be described in terms of the equipment on which they run (e.g. Mainframe, Mini and Micro computers). Instead it is becoming accepted industry practice to describe computing in terms of the use to which the computers and applications are put; namely, Corporate, Departmental or Personal computing, linked together by a Network. This is the basis of the CDPN model, used as a common reference model within ICL and with our customers (Fig. 1 identifies some of the main attributes and requirements for each of these types of Server).

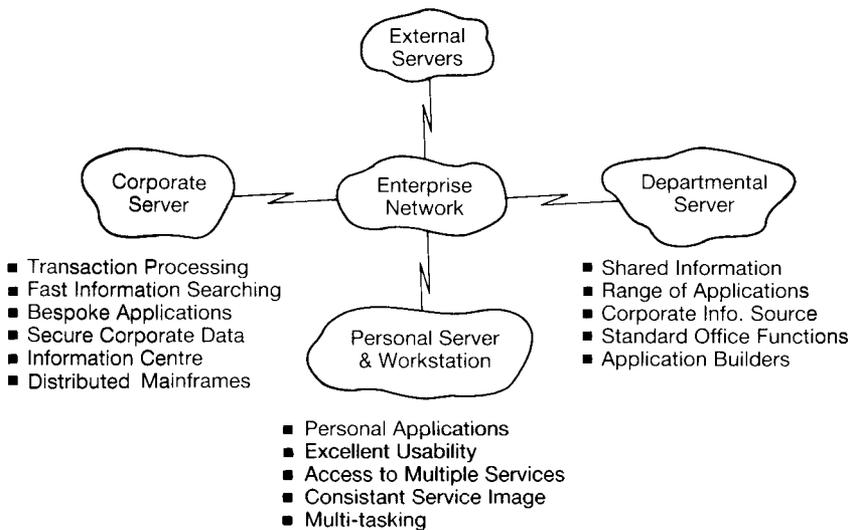


Fig. 1

In order to allow the construction of networked systems of any desired shape or complexity it is necessary to use modular “building blocks”, all of which must necessarily conform to the same basic standards. Because major corporate networks will contain systems from more than one vendor, these standards need to be defined and controlled by the industry, rather than being proprietary. Thus such standards are being widely adopted as corporate purchasing standards, used by organisations to ensure that, despite decentralised purchasing of information systems, all systems will still be able to talk to each other through the corporate network.

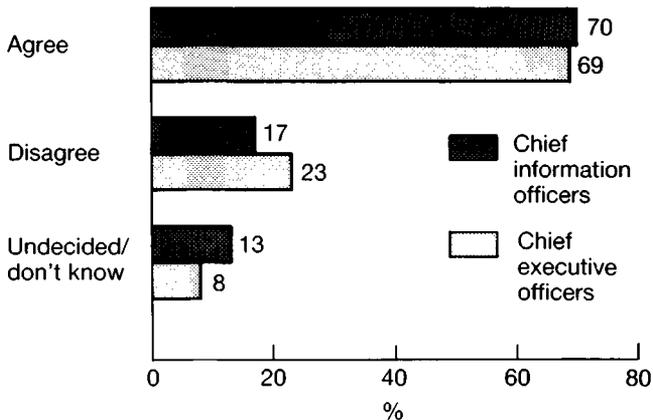
Having established its corporate network, an organisation will then need to connect it to the outside world. A growing dependence on public information providers and industry-specific networks, plus the need for inter-enterprise communication up or down the customer’s supply chain, means that a further, higher level of integration is necessary. The addition of multi-mode (e.g. voice, image) to this picture will add a further degree of complexity.

In today’s competitive world, a business needs to be managed as a single

cohesive entity and this can only be fully achieved if its information systems are fully integrated. It is also becoming accepted by the business community that IT is a prime source of competitive advantage, and as such is closely linked to the success of their companies (see Fig. 2).

Success and Strategic Information Systems

The Success of our company is closely linked to our ability to gain competitive advantage using information systems



Based on a survey of 286 chief executive officers and 591 chief information officers

Source: United Research Co. Inc. Morristown N.J. USA

Fig. 2

Consequently, business enterprises are increasingly likely to focus their in-house IT resources on areas of specific competitive advantage to their business. Thus they will want to buy-in standard solutions and commodity products for functional areas where they cannot gain significant competitive advantage; they will also want to subcontract much of the integration task. Thus an enterprise will typically place a number of contracts with separate suppliers in order to construct an IT business system. Management of these contracts to co-ordinated, successful completion is a highly skilled and demanding task which many will wish to devolve to a single prime contractor.

From the IT supplier's viewpoint, there are attractive business opportunities in managing the enormous implementation complexities of these contemporary IT solutions. Economies of scale can be gained by tailoring standard solutions, repackaging bespoke systems elements into standardised modules and systems, utilising standard building blocks for system and network construction, the sharing and cross fertilising of scarce and expensive skills, and the accumulation of knowledge and associated intellectual property rights.

The end result is that customers today do not in general just buy “boxes”, neither do they just buy “standard solutions”. Instead they are increasingly placing a range of contracts for the purchase of standard products, standard solutions, bespoke systems and major bespoke integrations through to facilities management. There are differences between industry types in what they will buy; there are also significant differences between customers in the same industry, depending on their needs and in-house skills.

So, overall the IT requirements and expectations of businesses are becoming more sophisticated and are thus placing demands of increasing complexity on their information systems and networks. Consequently, the need for delivery of fully integrated systems is paramount; so too is the need to integrate new systems with existing applications and networks. But the integration requirement does not stop there; a properly engineered enterprise-wide IT system should also fully integrate with the business organisation and its modus operandi. It should be structured and designed to full ergonomic standards so that it integrates socially and functionally with the business, its people and the way they work.

This requires far more than simply “plugging systems together” within networks. It addresses fundamental questions concerning the structure and shape of the business, relating this to how IT can assist any restructuring to gain competitive advantage.

4 The evolving demand for integration

Figure 3 attempts to capture the essence of the progressive evolution in market demands for integration. Over the years there has been a steady increase in demand for more sophisticated integration at successively higher logical levels; this is represented by the diagonal line showing integration level versus time.

Thus in the early days of computing, the integration task was essentially that of putting together a viable hardware structure. This soon extended to include software and applications, evolving through business systems to the current prime demand for business solutions, and leading on to the growing demand for enterprise-wide IT integration. In the further future this is likely to develop into large-scale IT integration between separate enterprises and, later, complete industries. A variant of this last item will be the changing of the boundaries between existing industries (e.g. Retail and Finance; Government privatisation programmes; Europe 1992).

The intrinsic difficulty of achieving the level of integration demanded, in a multi-vendor environment, at each point in time is indicated diagrammatically by superimposing the upper dotted curve which shows complexity factor versus integration level. The net difficulty of doing this using today's tools and standards is represented by the lower dotted curve.

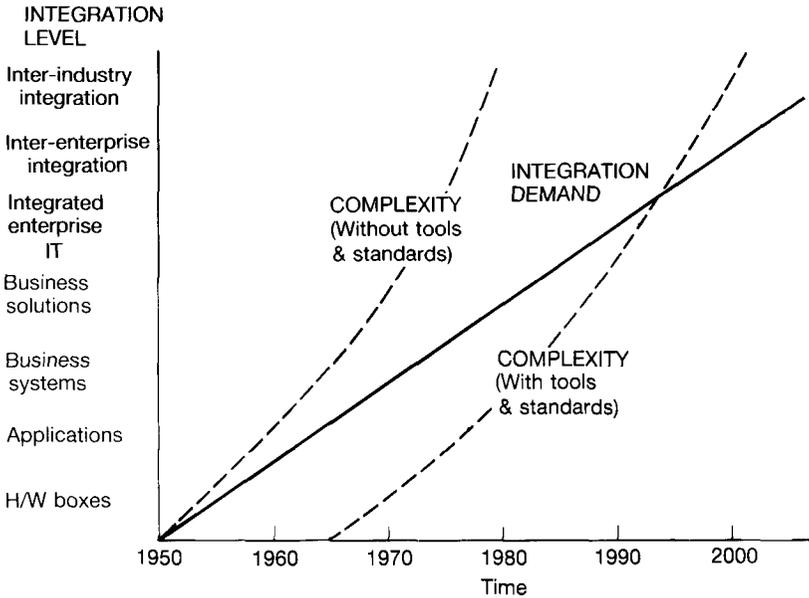


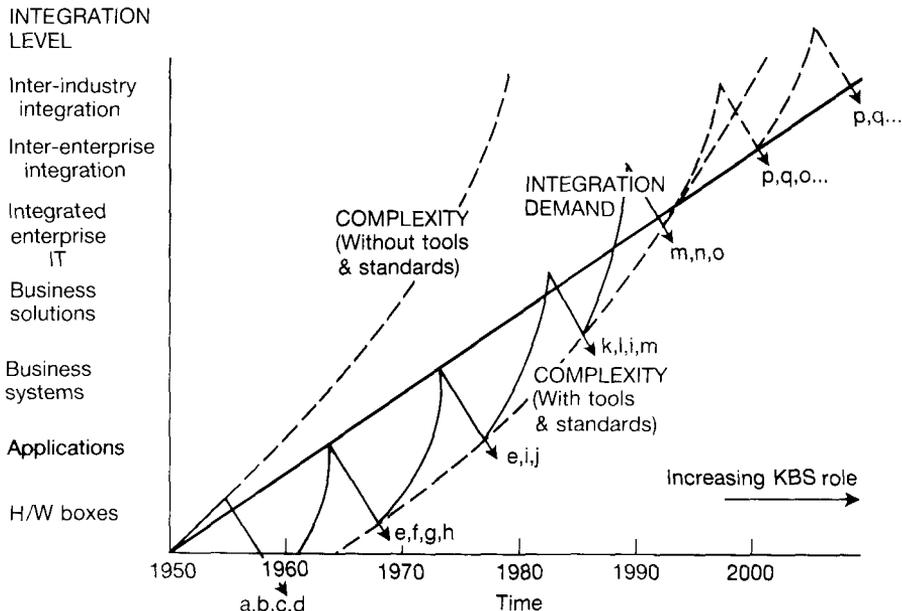
Fig. 3

This figure is representational and not definitive; it should be interpreted as follows: if a perpendicular to the x-axis intersects the complexity curve *below* the integration demand line, then the vertical distance between these two indicates the relative ease of achieving the integration level demanded at that time (the integration level is read from the y-axis at the point of the perpendicular's intersection with integration demand line). Conversely, an intersection with the complexity curve *above* the integration demand line denotes relative risk of failure or "degree of impossibility".

So, Fig. 3 illustrates that appropriate tools, standards, technology and techniques are essential to support all levels of integration activity. It is the role of Technical Strategy to point the way forward in these areas, so that the increasing complexities can be controlled and managed effectively.

Figure 4 indicates where some of the specific tools, standards and techniques of today contribute to reducing the intrinsic complexity of integration at each level and thus combine to bridge the gap between the two complexity curves shown in Fig. 3.

So, an appropriate range of tools and standards is, and will continue to be, essential to enable the IT industry to deliver the required level of integration. The industry still has a lot more work to do to facilitate the large scale integration between enterprises and industries (EDI is a start), and much remains to be done to improve the lower levels of integration. Knowledge



Key:

- | | |
|-------------------------------------|--------------------------------------|
| a: Design automation | j: Man machine interface (MMI) |
| b: Standard protocols | k: Human computer interaction (HCI) |
| c: Wiring standards | l: Community management |
| d: Interconnection standards | m: Security |
| e: Softwar engineering | n: Human factors in design |
| f: Applications standards | o: Integration architecture |
| g: Interworking standards | p: Electronic data interchange (EDI) |
| h: Interworking services | q: Open distributed processing (ODP) |
| i: Information Management standards | |

Fig. 4

based systems certainly have a major role to play from now on; likewise ODP appears to have much to offer in managing and integrating diversity.

5 Types of integration business

Within an IT supplier such as ICL, integration is something which must take place throughout the company in order to create, from available components and skills and processes, solutions which have sufficient value to its customers to bring profit to the company. The prime concern must be to deliver whatever components, systems and solutions bring the highest perceived value to the customer's business. Requirements vary widely between different customers and industries, so it is necessary to be able to transact different types of business, which need, in turn, to be supported by several different types of integration.

Customers also are integrators, buying from many sources to build their own

solutions. Different customers make widely different choices of what to integrate themselves and what to buy, depending on the value of the integration and their ability to afford and manage the necessary skills.

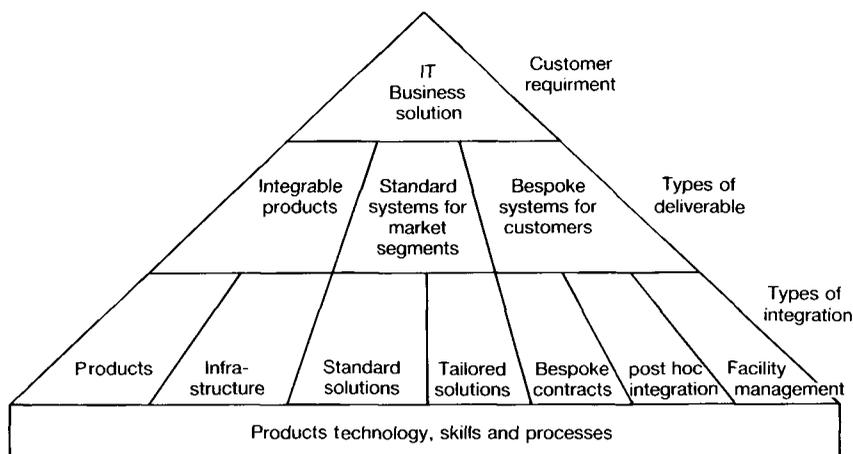


Fig. 5

Figure 5 depicts the various ways in which a customer's requirement may be satisfied, identifying three major types of business and seven main areas of supporting integration activity. These are:

- (i) The supply of a range of generic products (e.g. ICL's Networked Product Line), with common integration standards based on Open Standards, suitable for all markets and customers, that the customer himself integrates into his IT systems. This subdivides into products (hardware and software) and infrastructure (e.g. electronic office);
- (ii) the supply of standard solutions, designed for an identified market, integrated by the Systems Integrator from his generic products and industry-specific products, and optionally tailored as necessary to the needs of each individual country, sub-segment, or customer (for example, Retail in-store systems);
- (iii) fulfilling prime contracts for bespoke systems for an individual customer, integrated by the Systems Integrator and his subcontractors from products and technology of diverse origin, including the customer's existing installed system. This subdivides into three parts: Bespoke systems ("green field" developments), Post-hoc integration (integrating whatever the customer currently has installed), and Facilities Management. A formal Systems Integration contract will normally be based on one or more of these.

The core of the business of a full-function Systems Integrator is in being able to supply to its customers whichever combination of the above they perceive

to be of most value to their businesses. Full integration with the customer's business will be needed and this implies more than just plugging his systems together to form an Enterprise Network. The totality of the Enterprise system needs to integrate ergonomically with the business and its environment:

- organisationally: by providing access to the right *information* (rather than data) in the right form and by providing the correct functionality (sub)sets;
- socially: by being easy to understand, relate to and use, having consistent "seamless" interfaces and allowing people to work in the way they find most convenient;
- managerially: by being easy to monitor, change, update and maintain;
- visually: by having equipment styles and colour options which can be changed to blend with specific working environments in especially image-conscious markets.

So, to summarise:

- Every solution delivered to a customer is the product of a chain of integration functions (running from left to right in Fig. 5) of up to seven main types, each adding value. A given business unit within the vendor's organisation will be part of many such value chains and may sit at different points on each; the customer himself may also do some of the integration.
- Understanding and accurately quantifying these value chains and their complex interaction within the vendor's organisation, and their relationship to customers' own value chains will be essential for the vendor's ongoing market success and profitability.
- The fundamental requirement of an integration function is that it adds value rather than cost to a system or solution. It should not simply be there to "fix" repeatedly the problems which originate "upstream" in the integration chain. So the Quality process has a central part to play in Systems Integration now and in the future.

Within the ICL context, our Industry Marketing strategy will require us increasingly to be perceived by the market as being highly competent in all aspects of IT for our chosen industries. This requires us to be able to provide a complete range of IT services to those markets; in other words, to cover the full scope of functionality depicted in Fig. 5, together with the associated skills, processes, support and consultancy which this implies.

6 Channelling value to Customers

Figure 5 covers what needs to be delivered. However, there is a further major consideration; namely, by what means should it be delivered? Taking ICL as the model:

While ICL's objective is to be the preferred and trusted partner, and supplier

of value, to each of its customers, ICL cannot always be the supplier of all value.

However, each customer should have an IT Architecture that ICL understands and preferably supplies. This will be derived from mapping the customer's business IT needs, his Enterprise Model, onto a standard implementation architecture.

Within that IT architecture, ICL should seek to co-ordinate three major distribution channel elements under ICL management:

- the ICL Direct Sales Force;
- other product suppliers;
- other integrators.

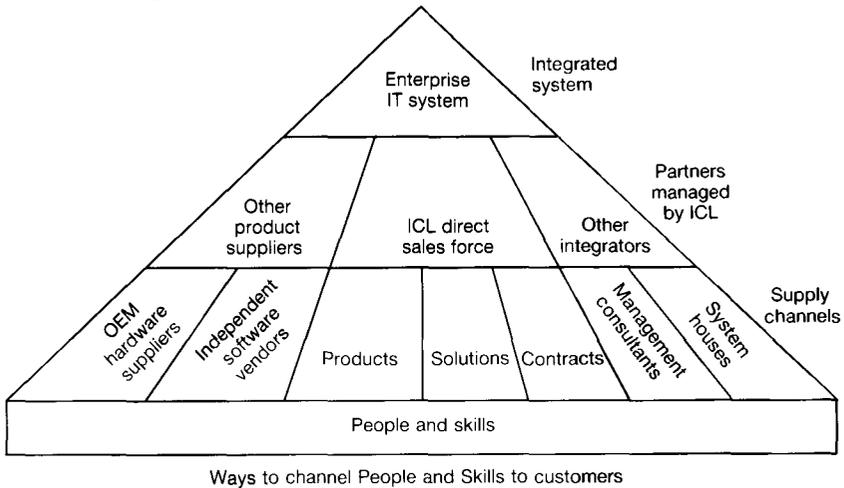


Fig. 6

Figure 6 represents the various channels through which people and skills may bring value to customers. The wide variety of customer requirements and the size and scope of contracts to be undertaken will require widely varying mixes of contribution between these partners. In some cases ICL may participate in consortia or undertake a specific part of a larger project.

The decision as to what is handled "in house" within ICL and what is devolved to partners will depend on such factors as strategic importance, skills available, profitability and local conditions. Any future acquisitions, collaborations and joint ventures will also have a significant influence on this decision.

7 Standards and Differentiation

Open Standards are an invaluable enabler for Systems Integration. They are good news for all the stake holders in the IT industry, both vendors and

users, because they are owned by the industry and are under public change control. Open Standards are a welcome sign of a maturing IT industry and a way of taming excessive, gratuitous diversity on the one hand and domination by one vendor on the other.

Two major issues are raised by the existence and growth of Open Standards:

How do vendors differentiate themselves one from another in an increasingly standardised IT world?

Can a line sensibly be drawn between that which is subject to Open Standards and that which remains proprietary (and thus highly diverse)?

7.1 Differentiation

The subject of differentiation in a technology-based industry needs to be viewed in the context of the evolution and maturity of the industry. This is because the balance between standards conformance and differentiation is dynamic; that is, it changes over time:

- In a *new industry* there are no precedents or standards to follow, so differentiation is primarily technological with a strong emphasis on (functional) novelty. There is no consensus on “the right way to do things” and there is not an obvious “winner” to imitate. The result is that everybody thinks they know best and so do their own thing.
- In a *semi-mature industry*, most of the technology will be “commodity” and will conform to standards. Conformance to higher level (e.g. functional) standards will gradually become mandatory. Differentiation will be mainly on additional core functionality and (a few) genuine technical innovations; all these will rapidly be copied by competitors. There will be a growing need to differentiate on other grounds such as building corporate or brand image, quality, creative packaging of offerings, market niching, service, etc.
- In a *mature industry*, almost everything will be “commodity”, including most of the technology, core functionality, quality, standards conformance, etc. Here the differentiation will tend to be on corporate image, fashion (easily copied, however), cost (not necessarily lowest), contrived (promotional) differences and second/third order functionality details. Very occasionally a major innovation may come along which changes the rules of the game and causes a transient (or, in extreme cases, terminal!) instability.

These are obviously generalisations which need to be interpreted differently for different types of industry, but I would summarise them as:

New industries differentiate on Technology

Semi-mature industries differentiate on Functionality

Mature industries differentiate on Corporate Identity

Currently the IT industry is semi-mature, but is moving towards the point where the leading customers are looking for more than functional differentiation of products, they want a company with which they can, with confidence, plan into the future. Of course such a company must be able to demonstrate it has sound technology and products and that it conforms to industry standards and that it can deliver a coherent, integrable range of products; but these will increasingly become mandatory qualifications for being a player in the market at all.

In the immediate future, there is still some scope for functional differentiation. This will mainly relate to macro “quality” issues such as:

- ergonomic design of hardware and total systems (i.e. “human factors”);
- manageability of networked systems;
- security of networked and distributed systems;
- accessibility of data across networked and distributed systems.

These will all, of course, have to cover multi-vendor environments and work within the framework of Open Standards.

However, these are likely to be transient differentiators, rapidly becoming “commodity” items as international standards move upstream to higher logical levels, and shared codes of “best common practice” become widely accepted in the industry.

Technology is likely to run further ahead of the industry’s ability to apply it profitably to business solutions. This will give further opportunities for technically based product and functional differentiation through innovative application of technology (e.g. Knowledge Based Systems is an area which clearly has great promise and untapped potential). However, we should not under-estimate the inertia of current systems investment which works to counterbalance radical innovation, despite the enthusiasm and fervour of committed pioneers.

Nevertheless, barring major discontinuities caused by somebody “changing the rules of the game” in the industry, one can anticipate a time not too distant when all these factors cease to be significant sources of advantage in the market, but instead become disqualifiers for those who can’t make the grade.

7.2 Scope of Open Standards

From the Systems Integrator’s viewpoint, there are great benefits in establishing international standards which are not controlled by any one supplier or national group. Their presence makes the integration task potentially less complex and, given that they are widely adopted as procurement standards, a “neutral space” is created within which vendors can compete on equitable terms. From the customer’s viewpoint, they do not tie themselves too closely

to any one supplier and can utilise an optimum mix of products spanning a variety of suppliers, confident that, because future products and systems will also conform, they can easily integrate future enhancements into their business systems.

The industry has made a notable achievement in establishing the current portfolio of intercommunication and interworking international standards and is already well down the track on standards of a higher logical level such as UNIX, CAE, Security and EDI. However defining and agreeing a standard is one thing; implementing it in a uniform and consistent way is something entirely different.

It is here that we start to see some of the flaws in this idyllic picture of a totally standardised IT world. For instance:

- Standards take a long time to formulate and agree and even longer to appear in a significant selection of products across the different vendors.
- Generally, the higher the logical level of the standard and the more complex its technical implications, then the longer it usually takes to formulate and agree, particularly where there is a clash of powerful vested interests and prior investment. (Paradoxically, the very high level standards such as EDI are relatively easy to agree; but it is the “middle ground”, into which much of the standards activity is migrating, which is the most intractable.)
- Detailed implementation variations of a given standard between different vendors mean that full compatibility and interworking cannot be guaranteed. This is a basic “engineering” problem of specification, tolerances, operating limits and conformance; information engineering has a long way to go to match the precision of mechanical engineering in this area.
- It is not usually possible to establish a standard in advance of at least some products being implemented. Thus there is a significant time lag between a market or technical opportunity first appearing and an appropriate standard being agreed. Because this time lag is so great, no major vendor can afford to delay development until the standard is agreed. Therefore, there will always be an accumulation of new developments using proprietary standards which run ahead of the international standards. As the complexity of international standards increases, this non-standard element will grow in size, creating “diversity-in-depth” which it will be increasingly difficult to align within the eventually agreed standards.
- Another driving factor will be the wish of vendors to build differentiation for their own products and systems on top of international standards. As higher level standards work encroaches on these areas, there may well be resistance to conformance, thus further delaying useful implementation of the standard.

Taking all these factors together, it will be necessary for the industry to define agreed domains and scopes for formal standards activity, and then to

determine a method for handling the integration of diversity in the areas thus excluded. There is no quick or easy answer to this problem, but answer it we must! Whatever the detailed resolution, the scope for technical differentiation will still be eroded, as discussed in section 7.1.

8 Who will succeed as a Systems Integrator?

Market success in Systems Integration requires a company to be perceived as achieving successful risk management on behalf of its clients.

Business success in Systems Integration requires a Systems Integration company to be successfully managing its own risks.

I have argued that Systems Integration is a label associated with the next evolutionary stage in the delivery of value by the IT industry to its customers. Therefore, the choice for an IT company such as ICL is either to become a systems integrator or a high volume supplier of commodity hardware and software (forseeing the eventual polarisation of the bulk of the industry's physical deliverables into commodity international standards conformant hardware "boxes" (e.g. UNIX engines), and de-facto standard applications and tools (e.g. INGRES)).

This begs the question of what sort of company is most likely to succeed as a Systems Integrator, bearing in mind that a supplier to customers in discrete vertical industry markets will need to transact a wide range of business (see Fig. 5) in order to be competitive.

What will it take to be a good, consistently successful Systems Integrator?

The short answer is "a lot"; the right culture, the right skills, the right products, the right organisation, the right image and the right associates. These issues are discussed in the following sub-sections:

8.1 Culture

I believe that the pivotal element is culture, and within that the commitment to Quality. "Right first time, on time, every time" seems to say it all; but not quite all. Quality will keep a company in business in the shorter term, but will not necessarily give it a future; some additional vital spark is needed.

This is, I believe, the role of Innovation. Like Quality, this is a culture which needs to pervade the entire organisation; everybody plays and it influences everything that is done. Innovation is all about finding better ways to do things and better things to do; about not wasting opportunities. It implies flexibility, acceptance of change, responsiveness, and the commitment to encourage, evaluate and, where appropriate, rapidly and fully implement new ideas. It does not mean just inventing new products, it applies equally to other areas such as organisational structure, processes, problem solving and

the way we do business; it is also proactive, rather than simply reacting to problems.

So, if Quality is “getting it right first time, on time, every time”, then Innovation is “doing it better each time”. In the wider marketing context, both deliver improved profitability but whereas Quality also delivers market acceptability, Innovation delivers competitive advantage.

I submit that the history of the Swiss watch industry over the last 20 years or so amply illustrates the distinction between Quality and Innovation.

Taken together these two really amount to professionalism, although this word may now be seen as a touch old fashioned. So,

QUALITY + INNOVATION = PROFESSIONALISM

because we expect more of the professional than just getting it right, we expect him also to contribute new ideas, approaches and insights.

8.2 Skills

Satisfying market demands is becoming increasingly services and people intensive, and requires ever greater levels of scarce and expensive expertise. Thus, the skills he can call upon are crucial for the Systems Integrator; they are his prime source of market differentiation and competitive advantage.

In high-profile SI contracts, the levels of skill, experience and expertise which go into identifying the opportunity, formulating the bid, negotiating the contract and selecting and managing the subcontractors need to be at least as great as those applied to the technical design and implementation issues. It is also all too easy at the bid stage to view the project simply as a vehicle for delivering tangibles such as hardware and software, and fail to consider the full value (and cost) of intangibles such as consultancy, training, IPR and risk factor. These intangibles have a real asset value and must be managed as such in terms of investment and pricing.

The IT industry should be able to learn a great deal here from other industries (e.g. civil engineering and defence) who have longer experience in bid and procurement management.

The evaluation of subcontractors and their products requires in-house expertise in the design, manufacture and development of hardware and software systems. How else is the Systems Integrator able to make valid assessments? These skills are, in any case, needed to produce “bespoke” elements of the solution.

In addition to bid, procurement and subcontractor management, strong project management is essential. Without this, even the best constructed contract will not achieve profitability.

World class technical specialists are also needed, but even more important are top-level “information engineers” who have a sufficiently wide view to ensure the viability and integrity of the total design.

Business modelling skills are also going to be increasingly important (see Enterprise Modelling in section 8.3) in the establishment of requirements.

8.3 Products

A successful integrator will be able to call upon a basic set of modular, reusable products developed within the framework of a unified architecture, and which conform to appropriate international standards. From these he will aim to construct a significant proportion of each delivered solution, the aim being to allow for the maximum diversity of solutions from the minimum set of modules. His profitability is largely dependent on how successfully he achieves this balance.

Every product must have a clearly defined role and position within the Architecture, be designed to be extendable and modifiable further down the integration chain, and be usable in many markets and applications. The product set will include integration tools which support downstream integration, both by the integrator and his clients.

The role of the architecture is to provide a coherent framework within which to construct networked industry solutions and integrate systems within an Open Systems environment.

The architecture and products will also need to allow for the real world diversity of proprietary standards and, progressively, new techniques and tools will need to be developed to accommodate this diversity within fully integrated Enterprise networks (see Section 7.2).

The above factors will form a significant proportion of the “economies of scale” which the Systems Integrator will have in comparison with individual customers. They do not mean that the Systems Integrator will always or exclusively use these products, or that they will all be developed in-house. The important thing is for a nucleus of “building blocks” to be available; this will be augmented or replaced by other vendors’ products as and when necessary to meet the client’s requirement.

Another essential capability is to understand fully, and agree with the client, his business requirements and how these will be used to measure the success of the IT business solution. Enterprise Modelling refers to a set of procedures and techniques which can be used jointly to explore with the client his business requirements and information flows, and to capture and record these. Such techniques can be used to identify how IT can facilitate organisational restructuring to improve the competitiveness of the business. Every Systems Integrator will need to have a set of tools to support Enterprise Modelling.

Developed to the ultimate, this could eventually constitute the first stage of a business solutions "full lifecycle management system", covering the entire IT infrastructure of an Enterprise.

This would require the integration of the tool sets for Enterprise Modelling, functional requirements modelling, Enterprise Architecture modelling, systems design engineering, development control, integration build control, integration installation control, operational management and maintenance.

Fully integrated systems require greater development, build and integration skills on the part of the supplier and have higher added value to the customer. Pricing should therefore be related to the value of the system to the customer, rather than "cost plus" from the supplier.

Arriving at a correct price requires thorough knowledge and analysis of "value added" factors at all stages of the supply chain, the sales channel and the customer/industry functional model. Tools to analyse these value chains will be essential to successful systems integration companies in the near future.

8.4 Organisation

The watchwords here are flexibility, adaptability and fast, responsive internal processes, with a toleration of differences in management ratios and financial controls within the organisation.

Internal accounting mechanisms need to recognise the different integration value chains within the organisation and apportion added value and added cost as appropriate.

Successful project teams need to be kept together rather than being disbanded at completion of their project. This reduces project start-up times and ensures a greater sense of identity and motivation.

There is likely to be an increasing market demand for systems which are perceived by the end customer to be bespoke to his individual needs, not just functionally but, in some markets, cosmetically (e.g. colour and visual design of equipment). This will require vendors to be able to handle, efficiently and quickly, short "bespoke manufacture to order" production runs.

Processes are also required to transform local products and bespoke project experience, developments and IPR into standardised products and reusable modules for use worldwide.

8.5 Image

The Systems Integrator will be doing business with major multi-nationals and other organisations who trade on a global basis, and must thus be the

sort of company with whom they can have total confidence and which can itself operate internationally.

A track record of successful delivery, high visibility, respect within the industry, and opinion-leader clients are all necessary qualities.

8.6 Associates

This recognises the fact that almost all formal Systems Integration projects will require the involvement of subcontractors, and that many will be big enough to require a consortium to be formed to make a bid. The latter will require the ability to collaborate constructively with companies who are normally competitors (cf. civil engineering).

So, for the Systems Integrator, having contacts, collaborations and agreements with the right people can also be considered as a source of competitive advantage.

Apart from this level of product and service collaboration, the full-function Systems Integrator will also need to collaborate in the field of research and development. Few companies will be large enough to do this all in-house.

One of the skills of the Systems Integrator will be in objectively assessing and selecting wisely his acquisitions, collaborators and partners.

9 Summary

This has been a fairly wide-ranging paper which has explored some of the major issues related to Systems Integration. The main points and conclusions may be summarised as follows:

Systems Integration is part of the continuing evolution of the IT industry in its delivery of value to its customers and the management of risk on their behalf. It involves additional complexity and additional risk for the supplier; the former being controlled technically by means of tools and standards and by access to the right technical skills, the latter being controlled by an appropriate range of management skills.

Systems Integration is more than simply "plugging things together"; it is an essential and pervasive part of the total "IT service" which major IT vendors will need to be able to offer to clients in order to satisfy their wide diversity of business requirements.

Existing IT suppliers ultimately will have the choice of becoming either Systems Integrators or very high volume suppliers of standard "commodity" hardware and software products.

To be successful, a Systems Integrator will need the right culture, skills,

products, organisation, image and associates; of these, an Innovation and Quality culture is the essential foundation.

The basis of differentiation between vendors in the IT industry will change as the industry matures. The prime sources of competitive advantage for a Systems Integrator will not be directly product or technically related.

The business of a full-function Systems integrator is resource and Services intensive, demanding high levels of skill and expertise which will be scarce and expensive in the market. It is also wide-ranging, requiring the learning of new skills and collaborations and partnerships with a broad spectrum of other companies. The limiting factor on business volumes will be the right skilled people, so the most successful companies will be those who can develop, or gain guaranteed access to, the best architecture, tools, products, techniques and processes to amplify the effectiveness of these people. Efficient and effective skills transfer processes and the right internal organisation and rewards system will also be needed to grow and retain the right skills in-house.

Finally, the Systems Integrator needs to respond to three major challenges:

The organisational challenge of running synergistically within the same company, three different types of business (products, standard solutions and bespoke), accommodating their diversity of business ratios, structure and metrics.

The business/technical challenge of how to systematize the volume delivery of variably-tailored versions of standard solutions, often multi-sourced, which achieve a “bespoke” level of fit with client businesses.

The management/technical challenge of succeeding as a prime contractor in formal, high-profile, state of the art Systems Integration contracts, and feeding experience so gained back into standard products.

10 Acknowledgments

A great deal of thought, discussion and argument within the SIS team provided the starting point and some of the material for this paper.

An architectural framework for systems

P. Henderson

Department of Electronics and Computer Science, University of Southampton

B. Warboys

Department of Computer Science, University of Manchester

1 System Integration Issues

Systems are built from components. Eventually established systems become components themselves in larger systems. Systems evolve by acquiring new components and by losing obsolete ones. The task of developing evolving systems becomes one of integrating existing systems with new components.

It is helpful in discussions of this sort not to distinguish between system and component, since ultimately every system is a component in some higher level system. Since we wish to concentrate on the way in which components interface with each other we will use the words *component* and *system* interchangeably in our discussions.

The key issue in system integration is evolution. Components exist to supply services. Existing assemblages of components are used to supply particular assemblages of services. As our requirements for services evolve, so the requirement to incorporate new components in an existing assemblage is conceived.

The big problem in system development is the need to evolve systems flexibly and economically. This means that we must be able to consider alternative solutions to a proposed evolutionary step. Then of course we need to be able to analyse the value of each alternative.

To these ends we introduce a language for system-level specification which is based on the notion of a component supplying a service and on the notion of components built from assemblages of subcomponents. This language is offered mainly as an illustration of the kind of language we feel is necessary to understand the structural problem of systems built from components. As yet we have not done the necessary experimental studies which will determine whether this language can tackle problems on the scale required by industrial practice.

We show how components can be combined and how the provision of services by the combined components can be calculated. On this basis we show how the hierarchical structure of a system and of a set of alternative

evolutions of that system can be described. We discuss the benefit of checking these structures for consistency of service provision. We also address the issue of the practical application of this technique which we claim leads to an obvious need for a *component catalogue*. Finally we discuss how the framework might be used as the basis for the evaluation of each alternative.

2 System Architecture

We often make use of the term *architecture* when discussing computer systems. Usually what we mean is, in some vague sense, the overall organisation of the components into a cooperating assemblage. Clearly for system building a good understanding of the architecture is very important. For example, most simple computers have the same architecture of processor, memory and other devices arranged on a bus. Understanding how these components interact and the rules which must be obeyed when combining them is fundamental knowledge for the designer of a new computer. Of course the designer may adopt a radically different architecture but, if his design is to be realised eventually as a working system, he will have observed some carefully determined rules for component design and component combination.

We take this as our definition of *architecture*. The architecture of a system has two parts

- the *constraints* which must be obeyed by a component to ensure that it is integrable
- the *rules of composition* which ensure that when components are taken together they also form an integrable component.

An example from software would be the architecture of a *purely functional program*. The constraints are that every function should be side-effect free. The rules of composition are that only function application is allowed. These are very simple rules to obey. It is remarkable that a wide range of software can actually be built within just these rules.

Another example might be *quasi functional programming* with the following architecture. Components are either side-effect free functions or procedures which are allowed side-effects. Two forms of composition are allowed. Application combines functions to form functions as in pure functional programming and sequencing combines procedures and functions to form procedures.

A third example arises when we consider the architecture necessary to exploit concurrent behaviour in a potentially parallel machine. This is the architecture of *shared abstract data types* in which we choose to arrange that the components are either processes or abstract data types. Abstract data types obey constraints which allow many processes to access them concurrently. The rules of composition include the protocol which must be obeyed by the processes to ensure important system properties such as freedom from deadlock.

The reason that an architecture is an important aspect of system design is that it gives us the means of overcoming the inherent complexity which goes along with size. It gives us the means to reason about the system as a whole and to ensure that, during its evolution, reasonable proximity to a valid implementation is maintained.

So any major system project can be expected to have an architecture for the system it is developing. This architecture may or may not be simple. It may or may not be explicit. It should be evident that the simpler and more explicit the architecture is, the higher will be the quality of the end product.

3 Framework Architecture

System Engineering for systems to be realised as networked combinations of hardware and software within a business environment requires engineers with a wide range of skills. The system engineer has to supply the solution which integrates the components of the system to profitably meet the business requirements. It is necessary for the System Engineer to take two distinct views of a system. Above him is the business need to be satisfied and below him is the technology with which he is expected to achieve it.

When looking upward toward the business, the system engineer uses his skills to

- be innovative about the market requirements
- analyse alternative solutions

When looking downward towards the technology, the system engineer is primarily concerned with the productivity within the engineering process which delivers the chosen solutions and he thus attends to

- tools, techniques and notations for describing the system
- the development and use of components which can be integrated

Occupying the middle ground as he does, the system engineer needs a *common language* with which he can talk both to the market strategist and the system developer. He needs to be able to describe the proposed architecture for his solution at a sufficiently high level of abstraction that a convincing argument can be made that the proposal does indeed solve the business problem. He also needs to be able to describe the proposed solution sufficiently precisely that the engineers to be charged with its realisation can be convinced that a workable solution is indeed being offered. In addition, it is necessary to be able to describe the solution as an evolution of an already established system with which this solution is to be integrated.

This paper is an attempt to evolve such a language. Necessarily the language will be at a very high level of abstraction. It will constitute a *framework* in which solutions to system problems can be evolved and planned. Because it accepts that systems are built from components which satisfy constraints and are combined according to rules then the language subsumes an aspect of

architecture. It tries to be non-committal about the details of the architecture, which components are hardware, which are software, which components are data, which are process and so on. Consequently we refer to it as the *Framework Architecture*. We expect it to be hospitable to all other architectures. It forms the basis for the highest level design. It is the most abstract description of a system.

It tries also to be *formal*. It addresses itself only to the structure of systems, what is built from what, rather than the structure of the processes which generate the system. In particular we are able to show a simple calculator style of manipulation of system structure descriptions. This formality allows us to be precise about the constraints and rules which must be obeyed when building or restructuring a system description. The constraints and rules together comprise the architecture. Our long term goal is to understand the relationship between the structure of the system and the process which produces it. This is the first step towards that goal.

In order to achieve this goal we need to address ourselves to the most abstract architecture which can describe system structure without predetermining the way in which the eventual system must be built. This we do in the remainder of the paper using a particular notion, of a component as provider of services.

4 Systems Description Language

To illustrate the basic model which we have defined we consider the services provided by a household electricity supply. This example has the advantage of being simple to describe and avoiding our prejudices about computer system design.

Assume that the ultimate set of services required by the end-user of the household electricity supply include light and heat. These services can be provided by a *lamp* and a *heater* which in turn require the services of power provided by a *ring* main.

Now we can define the system built by plugging the *lamp* into a socket in the *ring* by the expression

ring + lamp

Assuming that the *ring* has only a single socket we can not plug the heater into the ring unless we unplug the lamp, because the only socket has been used up. Suppose that we introduce a ring with two sockets *ring2* than we might expect that

ring2 + lamp

has a residual socket and that the residual socket can be used for the heater.

Both of the following should therefore be plausible systems

$(ring2 + lamp) + heater$

$(ring2 + heater) + lamp$

since we do not care in what order the appliances are plugged into the ring, only that eventually they are both so installed. Consequently, since + enjoys this property we choose to omit the parentheses and write

$ring2 + heater + lamp$

rather than the parenthesised form.

Consider the familiar two way extension lead. We might use it to extend our original ring main, with a single socket, to allow the lamp and the heater to be supplied. That is to say, the following is a valid system

$ring + twoway + lamp + heater$

To emphasise the structure we might choose to include parentheses

$(ring + twoway) + lamp + heater$

but the parentheses carry no meaning. We can write equations to note our knowledge about the equivalence of systems

$ring2 = ring + twoway$

Later we will see that we can give a precise meaning to equivalence between systems.

Well that all seems very trivial. It is of course. Our objective is to show that much more complex system structures can be described than those we have described so far, whilst retaining our desire to abstract as far as possible from detailed design concerns. Let us describe the structure of a house with two rooms, each supplied with a lamp and a heater. One possible description is

$house1 = room1 + room2 + ring2$

$room1 = lamp1 + heater1 + twoway1$

$room2 = lamp2 + heater2 + twoway2$

Another is

$house2 = appliances1 + appliances2 + ring4$

$appliances1 = lamp1 + heater1$

$appliances2 = lamp2 + heater2$

$ring4 = twoway1 + twoway2 + ring2$

With a little thought and a little substitution, we can probably convince ourselves that these two descriptions are equivalent. That is

$house1 = house2$

but the different sets of equations are suggestive of a different decomposition. In *house1* each *room* provides heat and light assuming we provide a single socket. We have taken the distribution of power between the *lamp* and the

heater to be the responsibility of the *room* subsystem. In *house2* we have chosen different subsystems. The *appliance* subsystems provide heat and light but expect the services of a pair of sockets. Since there are two *appliance* subsystems, we need four sockets which are provided by *ring4*.

All this structure is implicit. In the next section we will show how to make it explicit. Recall that we are involved in a process of capturing the structure of a system with little or no concern for the detail of its design. The system structure will be hierarchical and we will wish, when restructuring the description, to record this hierarchy explicitly so that we can clearly distinguish between alternative descriptions.

5 Hierarchical Structure

We introduce another system structuring operation. If we have a system built from (say) three subsystems *X*, *Y* and *Z* and these three subsystems are known respectively as *x*, *y* and *z*, then we denote this system by

$$[x : X, y : Y, z : Z]$$

This structure shows clearly the three separate subsystems from which it is built. Using the example from the previous section we can define *house1* as

$$[room1 : [...], room2 : [...], ring2 : [...]]$$

where the nested [...], describes the structure of each of the components. Where a component is primitive and has no structure we can describe it using the form []. So the structure for the entire *house1* is

$$\begin{aligned} & [room1 : [lamp1 : [], heater1 : [], twoway1 : []], \\ & room2 : [lamp2 : [], heater2 : [], twoway2 : []], \\ & ring2 : []] \end{aligned}$$

A rather better way to write this structure, especially as we can anticipate that such structures will become very large, is as a set of equations, as follows

$$\begin{aligned} House1 &= [room1 : Room1, room2 : Room2, ring2 : Ring2] \\ Room1 &= [lamp1 : Lamp1, heater1 : Heater1, twoway1 : Twoway1] \\ Room2 &= [lamp2 : Lamp2, heater2 : Heater2, twoway2 : Twoway2] \\ Ring2 &= [] \\ Lamp1 &= [] \\ Heater1 &= [] \\ Twoway1 &= [] \\ Lamp2 &= [] \\ Heater2 &= [] \\ Twoway2 &= [] \end{aligned}$$

The system which, in the previous section, we defined as *house1* is now defined by *House1* and retains explicitly the information about how it was constructed. In particular we see that the structure is a partitioning of the set

of primitive components. *House2* is a different partitioning of the same components with an obvious description which we omit here.

What is missing is any information about why these components are assembled into these particular subsystems. Of course our intuition about these particular components has led us to believe that this structure is sensible. It is because we consider that each component supplies some *service* which another requires. We have therefore to define the services which each component requires and those which it supplies. For example, taking the elementary components *lamp* and *ring*, we can say that *lamp* supplies the service *light* but requires the service *socket*. Similarly *ring* supplies the service *socket* but require the service *power*.

Let us define primitive components by the services which they respectively supply and require. We can write **requires socket supplies light** for the system which we have previously called a *lamp*. In general we will have a list of services which are required or supplied. Thus we have the definitions

lamp = **requires socket supplies light**
ring = **requires power supplies socket**

and from these we can calculate the definition, in terms of services, of *ring + lamp*. It seems reasonable to assume that this combination still requires the services which each separate component required, but which were not supplied by the other. Similarly the services supplied by the combination is whatever is supplied by either component and not required by the other. So we expect to conclude that

lamp + ring = **requires power supplies light**

where the service *socket* is hidden, because the requirement for it has been satisfied.

Now we can combine the hierarchical structure with the service information and arrive at a structure description which allows us to argue formally that the system has a sensible structure. We can complete the description of *House1* as follows

House1 = [*room1* : *Room1*, *room2* : *Room2*, *ring2* : *Ring2*]
Room1 = [*lamp1* : *Lamp1*, *heater1* : *Heater1*, *twoway1* : *Twoway1*]
Room2 = [*lamp2* : *Lamp2*, *heater2* : *Heater2*, *twoway2* : *Twoway2*]
Ring2 = **requires power supplies socket1, socket2**
Twoway1 = **requires socket1 supplies socket11, socket12**
Twoway2 = **requires socket2 supplies socket21, socket22**
Lamp1 = **requires socket11 supplies light1**
Lamp2 = **requires socket21 supplies light2**
Heater1 = **requires socket12 supplies heat1**
Heater2 = **requires socket22 supplies heat2**

Informally, the logic of this assembly should now be clear. Almost every service which is required by some component is supplied by a *nearby* component. The system as a whole provides heat and light in two rooms, but requires power to do so. Formally, we need to develop a calculus which allows us to manipulate and simplify the system descriptions. Although the concept of a service is a very abstract notion we should be able to specify constraints upon components in terms of the services which they respectively supply and require. The calculus should allow us to calculate for each system description the services which are at its interface.

6 A calculus of systems

The key operation which we need is a one which removes the structure we have so carefully recorded, for this will allow us to calculate the residual services unresolved by the particular combination which the description records. We will define an operation which *flattens* a structure by removing one level of the hierarchy, as follows

$$\mathbf{flat}[x:X, y:Y, z:Z] = X + Y + Z$$

So, for example since

$$\mathbf{Room1} = [\mathbf{lamp1} : \mathbf{Lamp1}, \mathbf{heater1} : \mathbf{Heater1}, \mathbf{tway1} : \mathbf{Tway1}]$$

we can calculate

$$\mathbf{flat} \mathbf{Room1} = \mathbf{Lamp1} + \mathbf{Heater1} + \mathbf{Tway1}$$

In addition, we know that

$$\mathbf{Tway1} = \mathbf{requires} \ \mathbf{socket1} \ \mathbf{supplies} \ \mathbf{socket11}, \ \mathbf{socket12}$$

$$\mathbf{Lamp1} = \mathbf{requires} \ \mathbf{socket11} \ \mathbf{supplies} \ \mathbf{light} \ \mathbf{1}$$

$$\mathbf{Heater1} = \mathbf{requires} \ \mathbf{socket12} \ \mathbf{supplies} \ \mathbf{heat} \ \mathbf{1}$$

So we can calculate

$$\mathbf{flat} \ \mathbf{Room1} = \mathbf{requires} \ \mathbf{socket1} \ \mathbf{supplies} \ \mathbf{heat1}, \ \mathbf{light1}$$

So what if we wish to flatten a more heterogeneous structure, such as *House1*? We need to extend the definition of +. We adopt the following rules

- 1 $[u:U, v:V] + [x:X, y:Y] = [u:U, v:V, x:X, y:Y]$ and similarly for different numbers of components
- 2 if **requires** *R* **supplies** *S* occurs as an operand of +, it can be treated as $[dummy: \mathbf{requires} \ R \ \mathbf{supplies} \ S]$

With these two rules we can calculate as follows

$$\begin{aligned} \mathbf{flat} \ \mathbf{House1} &= \mathbf{Room1} + \mathbf{Room2} + \mathbf{Ring2} = \\ &[\mathbf{lamp1} : \mathbf{Lamp1}, \mathbf{heater1} : \mathbf{Heater1}, \mathbf{tway1} : \mathbf{Tway1}] \\ &+ [\mathbf{lamp2} : \mathbf{Lamp2}, \mathbf{heater2} : \mathbf{Heater2}, \mathbf{tway2} : \mathbf{Tway2}] \\ &+ \mathbf{requires} \ \mathbf{power} \ \mathbf{supplies} \ \mathbf{socket1}, \ \mathbf{socket2} \end{aligned}$$

$$= [lamp1 : Lamp1, heater1 : Heater1, twoway1 : Twoway1, \\ lamp2 : Lamp2, heater2 : Heater2, twoway2 : Twoway2, \\ dummy:requires\ power\ supplies\ socket1, socket2]$$

This has removed one level of structure. A second level is removed if we calculate as follows

```
flat flat House1 =
  Lamp1 + Heater1 + Twoway1
  + Lamp2 + Heater2 + Twoway2
  + requires power supplies socket1, socket2
= requires socket11 supplies light1
  + requires socket12 supplies heat1
  + requires socket1 supplies socket11, socket12
  + requires socket21 supplies light2
  + requires socket22 supplies heat2
  + requires socket2 supplies socket21, socket22
  + requires power supplies socket1, socket2
= requires power supplies light1, heat1, light2, heat2
```

It is perhaps interesting to note that what we are actually doing in repeatedly revising the structure description is itself a system design activity. We are building abstract system descriptions by combining abstract components and are able to calculate properties of the assemblages which are supposedly desired properties. If they are not then we can revise our specification.

As far as it goes, this definition of flattening achieves what we want. Unfortunately it loses not only the structure, but also the knowledge about the intermediate services used by nested components. We shall reformulate the definition of primitive (or unstructured) components so that this does not happen.

Let us redefine primitive components so that the services which they respectively supply and require are represented by a graph. We can write $\{light1 \Rightarrow socket11\}$ for the system which we have previously called *Lamp1*. In general we will have a graph of services which are required or supplied, where the relationship recorded in the graph is *makes use of*. Thus we have the definitions

$$Lamp1 = \{light1 \Rightarrow socket11\}$$

$$Twoway1 = \{socket11 \Rightarrow socket1, socket12 \Rightarrow socket1\}$$

and from these we can calculate the definition, in terms of services, of *Lamp1 + Twoway1*, simply as the union of the two graphs.

$$Lamp1 + Twoway1 = \\ \{light1 \Rightarrow socket11, \\ socket11 \Rightarrow socket1, socket12 \Rightarrow socket1\}$$

Repeating the calculation for *House1*, we proceed as follows

$$\begin{aligned}
 \text{flat } \textit{House1} &= \textit{Room1} + \textit{Room2} + \textit{Ring2} = \\
 & \quad [\textit{lamp1} : \textit{Lamp1}, \textit{heater1} : \textit{Heater1}, \textit{tway1} : \textit{Twoway1}] \\
 & \quad + [\textit{lamp2} : \textit{Lamp2}, \textit{heater2} : \textit{Heater2}, \textit{tway2} : \textit{Twoway2}] \\
 & \quad + \{\textit{socket1} \Rightarrow \textit{power}, \textit{socket2} \Rightarrow \textit{power}\} \\
 &= [\textit{lamp1} : \textit{Lamp1}, \textit{heater1} : \textit{Heater1}, \textit{tway1} : \textit{Twoway1}, \\
 & \quad \textit{lamp2} : \textit{Lamp2}, \textit{heater2} : \textit{Heater2}, \textit{tway2} : \textit{Twoway2}, \\
 & \quad \textit{dummy} : \{\textit{socket1} \Rightarrow \textit{power}, \textit{socket2} \Rightarrow \textit{power}\}]
 \end{aligned}$$

This has removed one level of structure. A second level is removed if we calculate as follows

$$\begin{aligned}
 \text{flat flat } \textit{House1} &= \\
 & \quad \textit{Lamp1} + \textit{Heater1} + \textit{Twoway1} \\
 & \quad + \textit{Lamp2} + \textit{Heater2} + \textit{Twoway2} \\
 & \quad + \{\textit{socket1} \Rightarrow \textit{power}, \textit{socket2} \Rightarrow \textit{power}\} \\
 &= \{\textit{light1} \Rightarrow \textit{socket11}\} \\
 & \quad + \{\textit{heat1} \Rightarrow \textit{socket12}\} \\
 & \quad + \{\textit{socket11} \Rightarrow \textit{socket1}, \textit{socket12} \Rightarrow \textit{socket1}\} \\
 & \quad + \{\textit{light2} \Rightarrow \textit{socket21}\} \\
 & \quad + \{\textit{heat2} \Rightarrow \textit{socket22}\} \\
 & \quad + \{\textit{socket21} \Rightarrow \textit{socket2}, \textit{socket22} \Rightarrow \textit{socket2}\} \\
 & \quad + \{\textit{socket1} \Rightarrow \textit{power}, \textit{socket2} \Rightarrow \textit{power}\} \\
 &= \{\textit{light1} \Rightarrow \textit{socket11}, \textit{heat1} \Rightarrow \textit{socket12}, \\
 & \quad \textit{socket11} \Rightarrow \textit{socket1}, \textit{socket12} \Rightarrow \textit{socket1}, \\
 & \quad \textit{light2} \Rightarrow \textit{socket21}, \textit{heat2} \Rightarrow \textit{socket22}, \\
 & \quad \textit{socket21} \Rightarrow \textit{socket2}, \textit{socket22} \Rightarrow \textit{socket2}, \\
 & \quad \textit{socket1} \Rightarrow \textit{power}, \textit{socket2} \Rightarrow \textit{power}\}
 \end{aligned}$$

A final operation for adding structure to an unstructured component completes our calculus. We define $g \text{ part } m$ to partition a graph g with respect to a mapping m , where the mapping defines, for each node in the graph the partition to which it is assigned. This will enable us to restructure a description which has been obtained from flattening another description. So, for example

$$\begin{aligned}
 & \{\textit{light} \Rightarrow \textit{socket11}, \textit{socket11} \Rightarrow \textit{socket1}, \textit{socket12} \Rightarrow \textit{socket1}\} \\
 \text{part} & \quad \{\textit{light1} \rightarrow \textit{lamp1}, \textit{socket11} \rightarrow \textit{tway1}, \\
 & \quad \textit{socket12} \rightarrow \textit{tway1}\} \\
 &= [\textit{lamp1} : \{\textit{light1} \Rightarrow \textit{socket11}\}, \\
 & \quad \textit{tway1} : \{\textit{socket11} \Rightarrow \textit{socket1}, \textit{socket12} \Rightarrow \textit{socket1}\}]
 \end{aligned}$$

which is, perhaps more simply

$$[\textit{lamp1} : \textit{Lamp1}, \textit{tway1} : \textit{Twoway1}]$$

Clearly, with a little effort, and suitable mappings we can restructure **flat flat House1** to produce *House1* in a structured form. The mappings we use to accomplish this are the following

$$\{light1 \rightarrow lamp1, heat1 \rightarrow heater1, \\ socket11 \rightarrow twoway1, socket12 \rightarrow twoway1, \\ light2 \rightarrow lamp2, heat2 \rightarrow heater2, \\ socket21 \rightarrow twoway2, socket22 \rightarrow twoway2, \\ socket1 \rightarrow ring2, socket2 \rightarrow ring2\}$$

and

$$\{lamp1 \rightarrow room1, heater1 \rightarrow room1, \\ lamp2 \rightarrow room2, heater2 \rightarrow room2, \\ twoway1 \rightarrow room1, twoway2 \rightarrow room2, \\ ring2 \rightarrow dummy\}$$

If, instead of this second mapping, we use

$$\{lamp1 \rightarrow appliances1, heater1 \rightarrow appliances1, \\ lamp2 \rightarrow appliances2, heater2 \rightarrow appliances2, \\ twoway1 \rightarrow ring4, twoway2 \rightarrow ring4, \\ ring2 \rightarrow ring4\}$$

then we construct instead a structure equivalent to the alternative *House2*.

We have shown the basis for a language of systems. We call it *Framework Architecture*. It consists of some rules for combining systems to form systems and some constraints restricting what are considered valid systems. We can also specify derived constraints which we might require a system description to enjoy such as **flat** ($g \text{ part } m$) = g . A simple calculator has been prototyped in *me too* and used to check the consistency of the examples used in this paper.

7 A method for the analysis of systems

The way in which we see the *Framework Architecture* method being applied is as follows. Existing systems are analysed by constructing a description of each component in terms of the services which it supplies and requires. Using **part** we impose upon this the physical structure which the system currently enjoys. This leads to a refinement of the level of detail at which the components are described. Some services will be coalesced, because their separation contributes nothing to the description. Some services will be split in order better to describe the interfaces between higher level components.

Once an existing system is described in this way it is now possible to explore its potential for evolution and restructuring. Components can be removed and replaced with alternatives. Subsystems can be restructured, first by flattening them, then by restructuring them possibly with some alternative

components. A proposed evolution of the system can be studied by a comprehensive analysis of the extent of its disruption of the current structure. The potential for component reuse can be examined and the possibility of component exploitation can be demonstrated.

In practice many different descriptions of the structure of a system are likely to be held simultaneously, some for different versions of a system, some for analysis and planning. Management of a task of this size is a job for an IPSE. The calculator described in the previous section would then form a significant part of the system planner's toolkit embedded in that IPSE.

The knowledge about components can be captured in a catalog which enumerates the service requirements and service provision of each component. A component catalog lists these requirements and services for each component. It forms a database for both the consistency checking and for the value analysis which is used to choose among alternatives.

It is too early to say how practical it would be to establish a catalog of this sort. Some experimental work is required to determine how realistically components can be described using the notion of services.

8 Value Engineering

The structure of a proposed product can be evaluated also for the value which each component adds to the solution. There are a variety of ways in which this can be done. For instance we might assign cost and benefit values to each component, distinguishing between bought-in and internally developed solutions. We can go further and determine values for the retained skills. However it may be more appropriate to develop a dynamic model to supplement the static model described here.

To give a flavour of how value analysis can be accomplished assume that each service can be given a monetary value and each basic component a monetary cost. For each proposed solution we can calculate the value-added and the combined cost. If it is necessary to take other costs into account then they can be incorporated in the model as components of the solution.

By this means a series of alternative solutions can be evaluated to aid in the decision to choose between them. Again, much experimental work along these lines is required before the practicality of this technique can be assessed.

References

- 1 HENDERSON, P. and MINKOWITZ, C.J.: *The me too method of Software Design*, ICL Technical Journal, Vol 5, 4, 1987.
- 2 WARBOYS, B.: *CASE in the System Programming Environment – a case study*, CASEurope Conference, London, November 1988.
- 3 HERBERT, A.: *The Advanced Network Systems Architecture Project*, ICL Technical Journal, Vol 5, 1, 1986.

Twenty Years with Support Environments

Brian Warboys,

University of Manchester, Manchester M13 9PL, England

Philip Veasey

STC Corporate Information Systems, Feltham, TW13 7EJ

Abstract

This paper looks both backwards and forwards. It records some lessons learnt from living with Support Environments for nearly 20 years. It looks forward to our current objective of achieving an "active" support environment. It analyses the shortcomings and strengths of our original implementation in 1971 and outlines how they have influenced our thinking in the research and development of its successor the UK Alvey funded project IPSE 2.5. It examines the work underway to exploit the new technology now emerging from the IPSE 2.5 project. This work has highlighted the enormous organisational impacts of this new technology and some observations are made on the wider sociological implications of this style of PSE.

1 Introduction

The paper looking, as it does, both backwards and forwards, inevitably draws on personal experiences. As such it seems useful to include some introductory remarks concerning the systems we use as examples.

The CADES system was developed by ICL in the early 1970's as a Support Environment for the development of the ICL Mainframe Operating System VME (Virtual Machine Environment). It has been used continuously since then, accumulating on the way the development outputs of some 3000 man years of endeavour over 16 years.

The IPSE (Integrated Project Support Environment) 2.5 project is funded under the UK Government's Alvey programme and is a collaborative research project. It began in September 1985 and will formally terminate in September 1989. It is a collaboration between ICL, its parent company STC, Plessey Research (Roke Manor) Ltd., British Gas PLC, Dowty Defence and Air Systems Ltd., The University of Manchester and the Science and Engineering Research Council (Rutherford Appleton Laboratory). The ALVEY programme is concerned with improving the UK's capability in Information Technology and hence includes encouraging collaboration between Industry and Academia in the UK.

Since 1985, a group of Information Systems people with responsibility for providing ICL's internal systems have been directing work which has led to the development of SENSE (Support Environment for New Systems and Enhancements). This provides the internal, to ICL, software developers with an integrated set of methodologies covering both the management and technical aspects of their work. At the outset a review of CASE and particularly IPSE's was carried out. This led to a belief that they were in a special position to take advantage of the technology beginning to emerge from the IPSE 2.5 project and a strategy was adopted to ensure that the chances of doing so would be maximised¹¹. This included defining the processes in SENSE more formally than is the common practice for software development methods.

The work to create a SENSE IPSE is now being carried out through a programme managed by STC's Corporate Information Systems (CIS). The first part of the paper describes the concept of a Process Support Environment (PSE) which we believe can be achieved. The second part discusses the evolution of the technology used to support this PSE.

2 Process Support Environments

A "Process Support Environment" (PSE) is a support environment in which *process* is the key concept. Perhaps its most controversial quality is that it is "active" in two senses.

First of all the environment is active. This radically reduces the burden of the routine mechanics of following defined processes. Put simply, a work environment can become "active" when it has sufficient knowledge embedded in it for it to play an active part in the control of what happens. The word "control" here is important. In most environments where computers are helping humans in their work, that help is better characterised as "support".

To clarify this distinction, consider the example of an investment approval process being carried out in a conventional way, illustrated by the diagram below. The manager continually refers to his own memory, written procedures and possibly computer presented plans in order to determine what the next activity should be and what resources/tools can be used to carry it out. In the picture above, the manager directs that a cash flow analysis is the next activity and the human part of our resource then sits down with the prescribed spreadsheet tool which will support this work (see Fig. 1).

Compare this with the picture below in which the computer has taken over the routine job of deciding what the possible next activities are and of offering them to whomever the plan said should be doing them. At sign-on, our analyst is offered a choice of activities on which work can proceed and, on choosing the cash flow analysis (marked CFA in the diagram) is automatically presented with the tools, datafiles and "Help" text for the job (see Fig. 2).

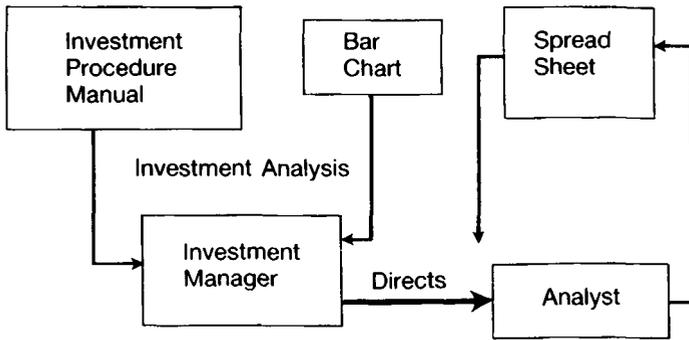


Fig. 1

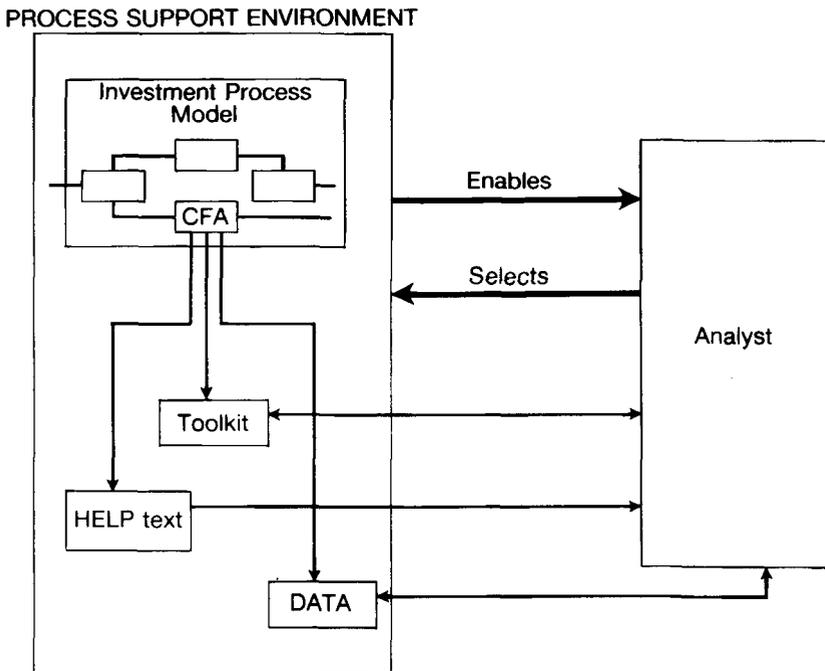


Fig. 2

The computerised work environment is now exercising a level of “control” over the process. The computer knows as soon as a job has been signed off as finished (since this can only be done through the computer) and is therefore able to offer the next jobs to whomever should be doing them. This situation also creates a completely new set of possibilities for automatic collection of information about what is happening or has happened.

Secondly, we can make the process active. By this we mean that the process can be changed in flight when appropriate. Indeed everyone involved in the process takes part in its determination by changing when desired those aspects of it for which they have change authority. They will, of course, only be able to make changes within the constraints determined by those with a higher authority.

2.1 What Types of Process?

CIS's immediate interest in the IPSE 2.5 technology is to use it to support software development. It is sufficiently general, however, to support many other kinds of process. Given its role, CIS has a strong interest in doing this so that PSE's are regarded as something which will support all processes which:

- involve many different human roles with complex interactions
- are not mass production

Thus the process of creating and managing a new product would be a suitable target for a PSE. For instance, if the product was a rubber band, where we would not be interested in each separate instance of the product, we would not create a PSE for its manufacturing process though we might do so for its design. PSE's will be useful for processes where each instance could be, and sometimes must be, regarded as a separate project.

2.2 Support Down to What Level?

In order to answer this question it is necessary to appeal to what is hopefully a fairly common understanding of the way in which activity in processes can be modelled by networks. The diagram below shows part of the activity network for a project process and indicates how we can separate out from this the activities which are carried out by a particular role. A role is characterised by its entitlement to perform certain tasks, its rights to the resources necessary to carry them out and its relationship to other roles. Examples of roles could be "designer", "project manager" and "quality assurance." We would expect to use the PSE technology support to create support for the process down to, but not including, the level of a specific activity within a specific role. The technology would also be used to call the support required for the specific activity. This support would not generally use the PSE technology (though it might) but the user need not be aware of this. Thus, from a users point of view, the PSE could be delivering support for the whole process at all levels (see Fig. 3).

2.3 The achievement of an Active Environment

To understand what will be required to create an active environment it is necessary to look at the process cycle depicted in the following diagram which shows the steps which occur with or without a PSE. The diagram is largely self-explanatory. A method is expressed in general terms applying to

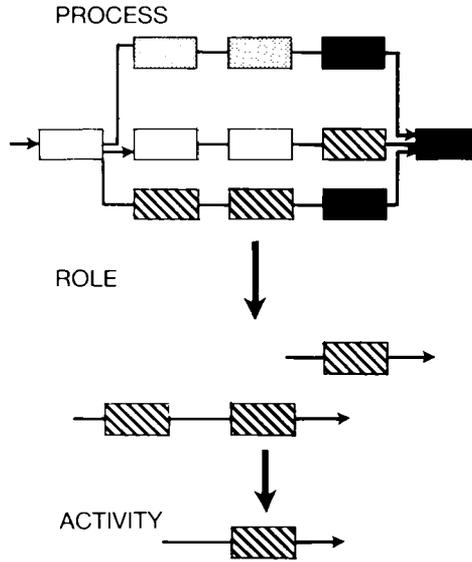


Fig. 3

all instances of the use of the process. Choosing it may simply involve following an existing procedure or may involve considerable creative effort. For example our method may be for the development of any product, so that if we move down to the first level plan this now describes the activities and resources required to create the particular product we are interested in. Estimates of the time required for the activities, and the amounts of the resources, are then applied to the plan which makes it possible to calculate schedules. Once a viable schedule is derived the plan can be implemented by committing real resources. It can then be executed by carrying the work out. Monitoring of the process provides information which can be fed back to management who can evaluate whether there is any need to change the plan at any of the three levels (see Fig. 4).

For an active environment the implementation step must also involve the creation of a machine-executable model of the process. The fully defined plan can contain all the information required to do this. Each step in the execution of the process is then controlled by the machine driven environment. In the "passive" environments to which we are accustomed any correspondence between plan and reality relies on human action totally – with results which are frequently disappointing.

2.4 User Requirement View of a PSE

Figure 5 is currently being used as the starting point for a top down user requirement specification for the PSE. The area labelled Process Control

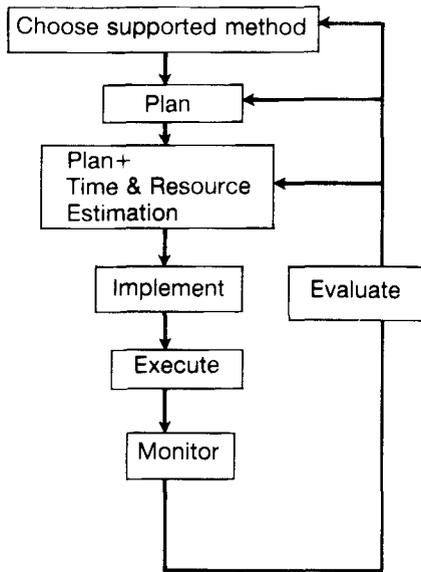


Fig. 4

indicates those parts of the total environment which are specific to the particular process which the PSE has been designed to support. The environment provides simultaneous support of more than one instance of a particular role type, e.g. several designers. Each human role player is wrapped around with:

- personal computing capabilities
- wordprocessing, spreadsheet etc. - electronic mail - access to external information and information systems
- support for the current activity
- tools (which may themselves be major software systems) giving automation and automatic adherence to many product quality standards - expert systems - context dependent textual help
- process control; giving automatic adherence to the process being supported.

The roles interact by passing objects (inputs and outputs of activities) between them, under process control, via a shared information store. The objects must be allowed to take many forms, e.g. formatted data, text, diagrams.

For this reason an object oriented approach appears suitable. The requirement for simultaneous access by multiple users will be met by using PISA¹⁰ (Persistent Information Space Architecture) technology developed by another Alvey project.

PROCESS SUPPORT ENVIRONMENT

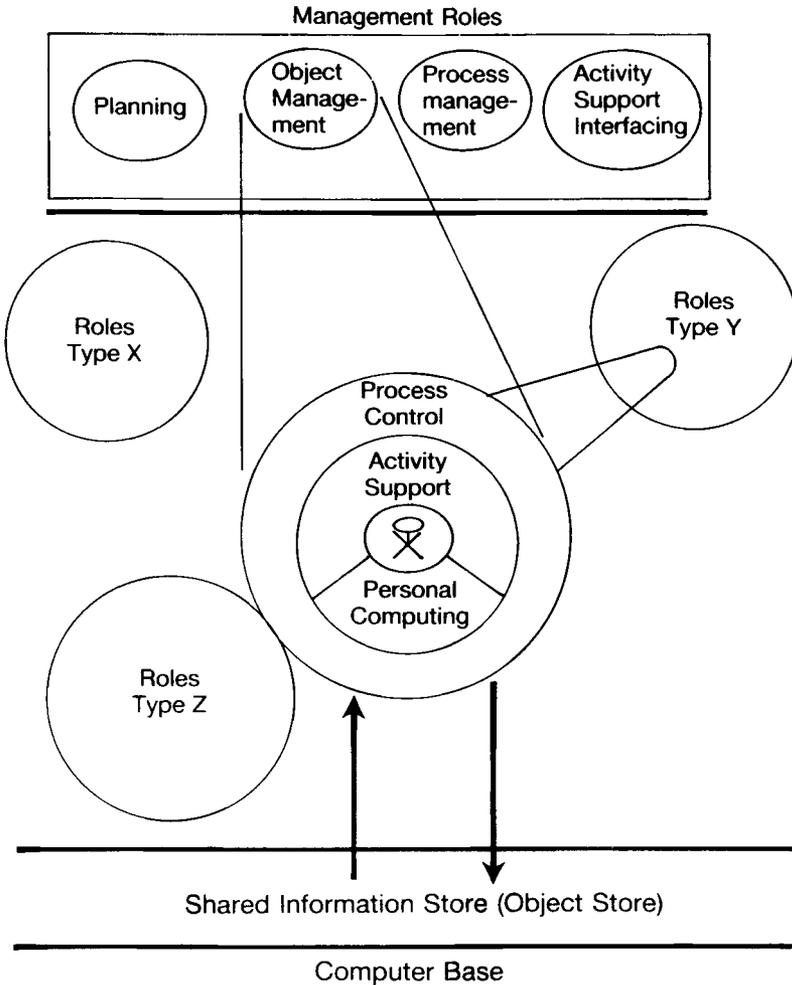


Fig. 5

In addition to the roles of the process being supported, the PSE must host the following management roles:

- planning: this allows scheduling of the activities and resources and evaluation of progress. Plans are automatically updated using data collected automatically during the execution of the process
- object management (corresponding to many people's use of the term configuration management)
- version and variant control - build control - access control - fault reporting and change control

- process management
- creation and change of process. The requirement for easy process change implies a need for a suitable language for defining processes – assessment of impact of process change
- activity support interfacing. This must enable easy and rapid change of tools with different levels of integration into the PSE. Very low levels of integration should be available with corresponding low integration costs. This will ensure that we do not deny ourselves the chance of experimenting with the increasing profusion of new and potentially productive tools
- attaching tools and expert systems – providing help text

Full definition of and most of the support for these management roles will be common to all PSE's

Other distinguishing characteristics are:

- embodiment of a full understanding of the interactions between the different roles so that different types of activity are integrated. This is achieved through the executing process model
- it focuses on providing support and control for human roles rather than carrying them out automatically. However, the human roles are co-ordinated with roles which have been fully automated
- it allows easy and rapid change of process. This applies to both standard processes, which will be continually evolving, and to a process which is currently executing. Thus the process is exactly what is wanted, and is feasible at the time
- as the process is carried out, it automatically collects data which can be used to improve the process and provide better estimates in the future.

3 Origins of the Technology

In order to understand the reasoning which has led to our current approach to Support Environments it is necessary to examine our experiences of development under the influence of such tools. In spite of recent hype which suggests that only recently has the industry begun to invest in any form of integrated approach to development support we, (at ICL in the Mainframe Division) at least, have been using such a support environment for nearly 2 decades. The environment in question is known as the CADES^{1,2} system and has been used since 1972 for the support of the development of the ICL Operating System VME³.

This system accurately reflects the industrial concern prevalent at the back end of the 1960's. Focus on these concerns is due in no small measure to the two successful NATO Software Engineering conferences of 1968/69^{4,5} which are arguably responsible for the very term Software Engineering.

Examination of our experiences with this system over the last 16 years led to some interesting observations on both what has changed and unfortunately what has not during this period. The observations fall into two categories:

- the industrial scenario and development concerns of that era
- the technology developed to address those concerns.

Perhaps more than anything they are timely reminders at this time, when there is an unseemly rush to standardise Support Environments, that our engineering expertise is still far from being mature. They show us how much has changed (and has still to change) in even our understanding of the software development process. This must lead to concern that in the realisation of the need for an “OPEN” standard we should not hastily define a very closed system.

4 Our Industrial Scenario in the early '70's

Prior to the early '70's ICL (in its various early forms, English Electric, ICT etc.) had produced a number of comparatively small operating systems and associated compilers for the relatively small mainframe machines which existed at the time.

These systems were produced as discrete components and as they gradually grew larger the approach was essentially to treat the enlarged components as collections of programs and farm out the development of the programs to separate groups of programmers. We found that this approach was inadequate and indeed that the cost of the systems compared to earlier systems increased exponentially rather than linearly with their increase in size.

Thus in 1971 when ICL set out to produce a major operating system, VME, for its new range of Mainframes (what became the 2900 and the Series 39 ranges) it recognised that it had a major task on its hands since all the indicators were that the development would never finish. In a sense this has proven to be the case since some 200 development staff have been continually developing the system since that time. The result was to establish the CADES project to develop a rigorous system development methodology, Structural Modelling, and a Computer Aided Development and Evaluation System, CADES, to support the methodology.

4.1 The perceived problem circa 1970

The problem was perceived as being product related. Early documents² state: “The large development team would need to be able to identify and preserve the overall structure of the operating system. Experience on earlier systems had shown how difficult it was to protect a large system from structural decay. The team would need to be able to distinguish between features which affected the overall structure of the system and those which were merely cosmetic.”

The methodology and computer aided system would have to facilitate all stages of the operating system development process, i.e. high level design, low level design of implementation, construction, system generation and mainte-

nance. They would have to encourage the codes of good practice which prevailed within the computer industry, i.e. structured programming, data/entity driven design, delayed fixing and binding, design for resilience etc.

4.2 The solution

These problems led to the introduction of the formal development methodology known as structural modelling and to the creation of the CADES system to provide project support for it (a PSE!). Structural modelling was first described in detail in¹ in 1973. It followed the philosophy that system development should be primarily a data-driven top-down process. (We had read the NATO conference reports^{4,5!}). The emphasis was on the modular structure of the system being designed to manipulate defined data items, rather than data items being invented to support the encoding of abstract functional requirements. Each code entity was termed a holon, a term borrowed from Koestler⁹ to describe utilities in an hierarchic system.

A language called SDL (System Development Language) was devised to allow the expressions of holon-holon and holon-data relationships. SDL was also used to describe how each holon uses its relationship with other holons and with data items in order to carry out its particular functions. The hierarchy of SDL holons represents a gradual refinement of the total description of the system, each level being a complete description of the system at that level of design, the level being fixed by the accompanying data tree decomposition. At the lowest levels this use of SDL merges completely into the implementation programming language S3, a close relative of the procedural language Algol68. The CADES database was set up to record information about the various holon-holon, holon-data relationship and this was then used as the basis for version control and other management support purposes. It had become, of course, a Support Environment (see Figs. 6 and 7).

5 Properties of its successor IPSE 2.5

5.1 Observations on the CADES Approach

Within the size and scope of this paper we can only attempt a very superficial analysis of the approach but the more significant highlights are:

- (a) The approach was product structure derived. This leads to some good and bad attributes. The best is perhaps that the architecture of the system would play an active role in the subsequent design decomposition. The theology of the product approach was reflected directly in the tools used for development. Thus both the architecture and the tools system have survived the test of time. The same system is still used 16 years and some 3–4000 man years later. It has demonstrated that Support Environments can have longevity. The worst attribute is that the process of development was necessarily made subservient and thus although the product has not suffered structural decay the process of

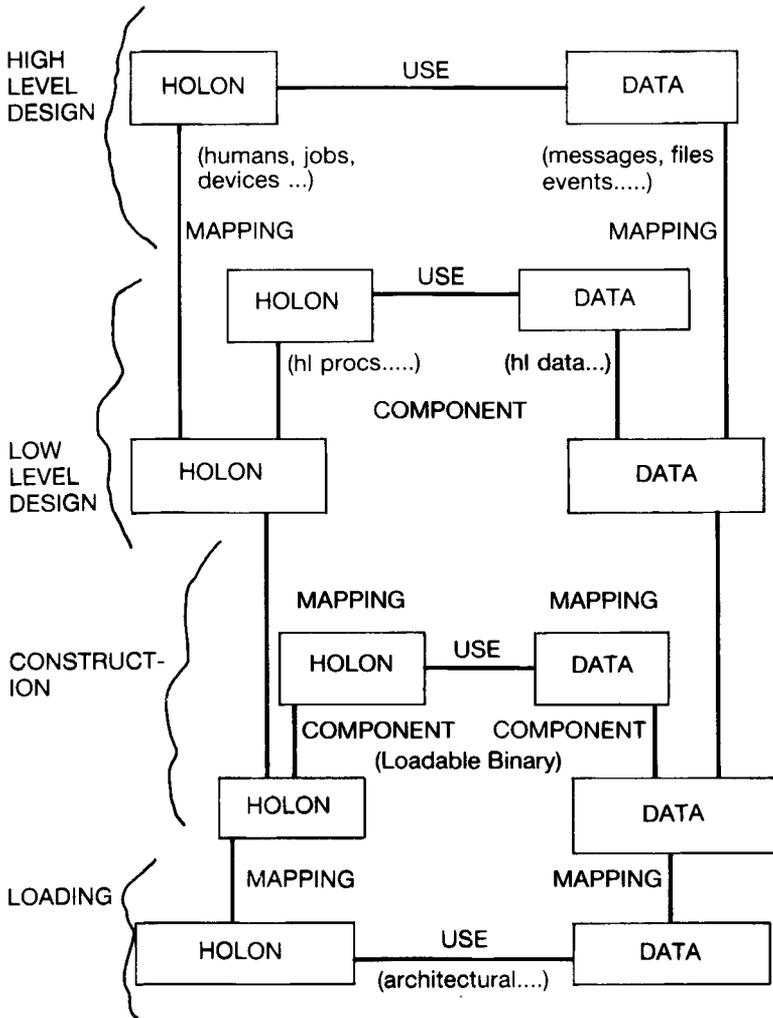


Fig. 6

development has not been able to adapt properly to more modern influences. It is particularly noticeable that the “total life cycle” outlined above made no mention of specification. Of even more importance was that the granularity of the database was product oriented and thus the granularity of the supported process was of necessity constrained to the level of the product modularity. Management of the process was and is conducted at the holon level. Re-use, optimisation, version management and all the other engineering concerns were thus also constrained to this level.

(b) The approach was essentially one of “Design then Evaluate”. Support

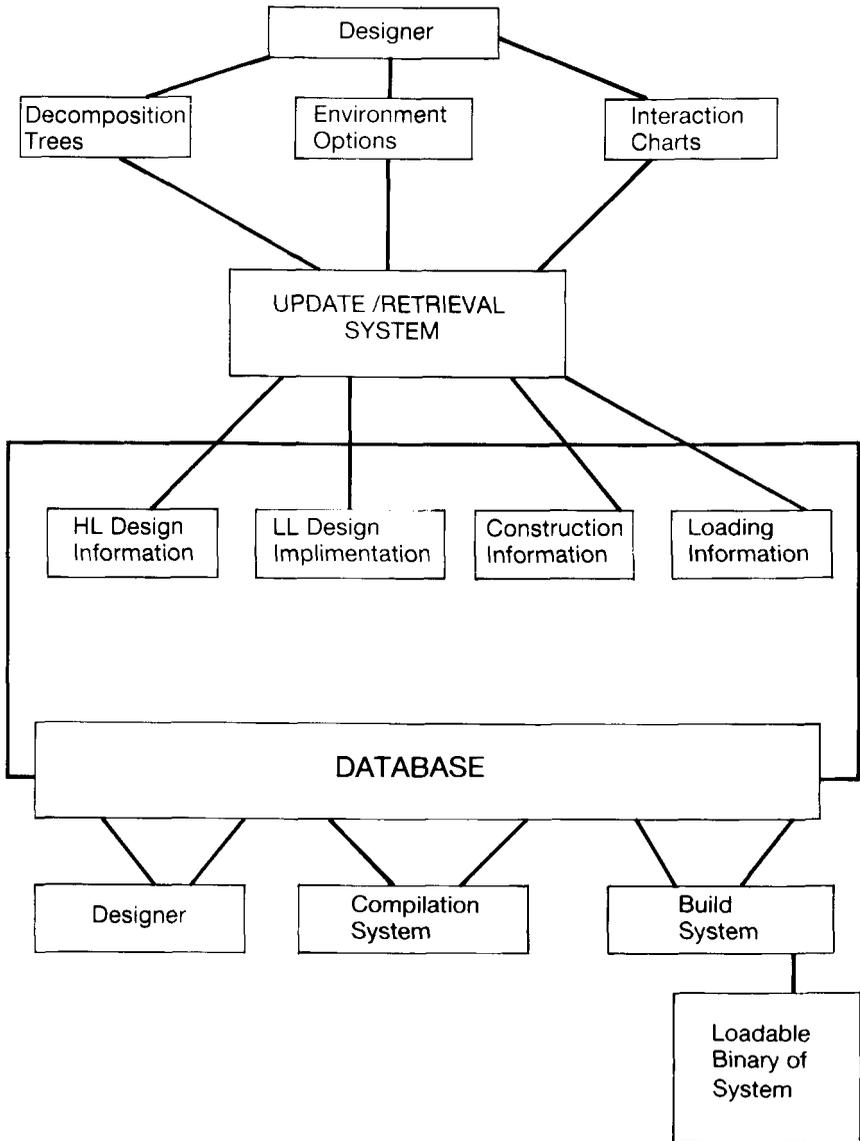


Fig. 7

for interaction was provided but essentially the system is oriented towards design capture of new modified functions and use of conventional (Codasyl based) database technology to invert and evaluate that design. The notion of prototyping was missing and the ability to reason about the design was restricted to structural analysis. Again the primary goal of managing the avoidance of product structural decay was realised

but the granularity of design entities essentially restricted the ability to reason except at holon-holon level.

- (c) The system was a closed one. Not really in the sense of current Open Systems concerns, since one could easily define a "Public Tools Interface" to the system, but in the sense that any tool added to the system was constrained by the core style of schema representation. New tools had to be totally integrated and the resultant costs severely restricted the ability to experiment with new toolsets, to rapidly discard and acquire new tools and generally to adopt a flexible tools strategy.
- (d) Many mundane issues are of fundamental importance. In the end most effort was expended on Version Control (also referred to as Configuration Management) and it is clear that is of paramount importance that such issues are addressed in such a way that they are all embracing and transparent to tools providers.

5.2 Subsequent conclusions and influence on the IPSE 2.5 approach

- (a) It is clear from the above that the principal constraint to a flexible support environment is, not surprisingly, the basic architecture of that environment. The decision on granularity had an all-embracing impact on both the approach to an open tools policy and to the type of development process which the tools enabled (and/or supported). In fact the decision to base the environment core on the granularity of the entities to be developed was the key factor. Our experience is that any attempt to base the Support Environment on the entities to be developed rather than the process to be used for the development is doomed to the same constraints as the CADES system imposed. How many of the current CASE/IPSE approaches are following this path? Clearly a major influence on the IPSE 2.5 project was the desire to build a Support Environment with a Process Control Engine as its core rather than an operating system style core with an interface fashioned primarily to entity manipulation.
- (b) Our Design then Evaluate strategy of CADES had highlighted the constraints on better approaches to development. In particular we view the need to reason about our specifications, designs, implementations as being fundamental to good software development. This ability to reason implies a level of integration of process and tools which needs to be determined by the nature of the interaction between the reasoner and the tools and not by the product modularity constraints. As workstations increase in power and in screen and 'tactile' support so the notion of a reasoning assistance will become increasingly tractable⁷. However it is also clear that the Environment is still required to support the conventional level of granularity as represented in the CADES system.

The second major influence on the IPSE 2.5 project was therefore the desire to investigate the integration of these coarse and fine grain support options. There were also implications on the Interface in terms of the management of vast quantities of fine-grain entities and their

- abstraction into coarser-grain entities, for example those of management interest such as plan impacts.
- (c) The closed nature of CADES had highlighted the constraints which such an architecture placed on flexible tool acquisition and disposition. Increasingly there is a need for flexibility in creating hybrid design processes and this implies flexibility in levels of tool integration. A simplistic "Operating System" style CASE will not enable the rapid construction of heterogeneous tools systems. OPEN systems imply a speed of change as well as community-wide ownership. It is important to ensure that the community-wide ownership desire does not inflict upon us a closed system in terms of development paradigms. Thus another major influence was the desire for the process based core to be the means of flexible and very cheap alien tools interworking, a generic "meta" Public Tools Interface. Thus we consider of vital importance to an Open CASE system the provision of a standard means of construction of Public Tools Interfaces rather than any one Interface *per se*.
- (d) The notion of Component Re-use had been an early motivator in the CADES system. The hope was that by the use of Database support we could develop a generic approach to "Libraries". Indeed that commonality would be established at many levels. In practice this did not happen. The means of specification and then subsequent browsing were not available, but again, even if they had been, the fine-grain integration which such "pattern-matching" implies would not have been realised by an "entity"-based core. Again some support for the recognition of the development process in order to enable re-use was required. It was also clear that the development process was such that the re-use of process fragments was as great a contributor to productivity as any design component re-use. CADES did not aid the long period of gestation concerned with the construction of a process, in many cases a heterogeneous assemblage of sub-processes, to solve large scale problems. This need to employ a variety of solution strategies in any given system development is, we believe, of great and growing importance.

5.3 The IPSE 2.5 Approach

The purpose of this kind of Support Environment is to provide the means by which the process of developing, maintaining, supporting and enhancing information systems is made more efficient, in both quality and productivity terms. Traditionally such Environments, be they for the support of programmers or the support of projects, have been considered as tools to support people who have tasks to carry out. The view taken in the IPSE 2.5 project is to stand back from this position of "users and tools" and consider the problem as a whole.

The process of developing, installing and changing information systems is one which involves the co-operative efforts of many people. People are involved in this process because of the intellectual nature of the various tasks.

We should not forget that no tool, as yet, can remove the essential involvement of the human being. As Dijkstra⁸ said in 1972, "We shall do a much better programming job ... provided we respect the intrinsic limitation of the human mind".

The IPSE is thus a means of supporting the whole process rather than being a collection of tools which assist particular activities or classes of activities within that process.

The essence of the integration component of IPSE 2.5 is based on the notion already expressed that an IPSE is about supporting the process of systems development, a process in which people (the "users" of the IPSE) play a very significant part. In many ways this is the logical successor to systems such as CADES where the environment is seen as providing the components out of which the process is formed, but in a completely general way which is ignorant of process.

5.3.1 The Process Control Engine (PCE) At the heart of an IPSE 2.5 system is something called a Process Control Engine (or PCE). The idea is that the PCE is a computer system which provides the (changing) working environments for its users, the people involved in the development process. The PCE is cognisant of the process itself and is thus able to provide the appropriate working environments at the appropriate times. The means by which this is done (and how much can be done) is the essence of the IPSE 2.5 project from a technical point of view, taking particular account of the requirements to support the use of formal methods and in particular formal reasoning.

5.3.2 The Process Modelling Language (PML) It would be possible (but not useful) to build a PCE which only supported a single type of development process. More usefully, it is necessary to provide some means by which a general purpose PCE can be "programmed" to support different processes. In reality, such processes are an amalgam of many separate processes or maybe just constraints on how particular tasks may be carried out. ("I don't care how you do it, but do it by Tuesday!"). The concept of a Process Modelling Language (PML) is introduced to be the means by which such different process fragments can be described and composed. Such fragments also provide the means of "alien" tool interworking. The working environments to be provided by the PCE for its users must include all the things that a user "says" he requires for the job in hand. The PML must therefore provide for the descriptions of objects upon which the user might carry out some operations, tools to help him carry out those operations, and means of communicating changes to the working environments of himself or of others. These are minimum requirements of the PML but serve here to illustrate the concept. In "traditional" Support Environments, the database schema (or rather the language in which it is expressed) can be seen as an embryonic PML, often augmented by the command language of the host operating system. Figure 8 shows the relationship between a user (i.e. a person involved

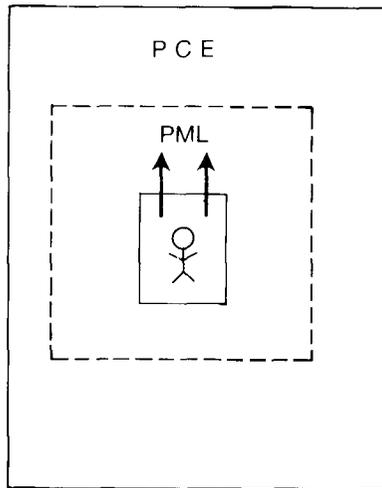


Fig. 8

within an overall development process), PML and PCE. The type of interaction that the user has with the system is determined by the PML description. The PCE is an engine which supports these descriptions. It is of note that the approach being taken is essentially that of explicit process descriptions. It is possible to imagine an approach by which a model is "learned" implicitly by the PCE from actions carried out by users, such models then being available subsequently. We believe that the interactive nature of the people/tools society which the IPSE encapsulates calls for a more explicit approach.

It is however worth noting that explicit models are the basis of "what if" type analyses (or more general property analysis) as well as being used as described here. This idea is one being pursued by the project in particular as part of its work in providing support for project management.

6 The relevance of the IPSE 2.5 approach to Software Engineering processes

To understand the relevance of such a generic approach rather than the stylised "operating system" core beloved of many CASE offerings there is a need to examine the current context within which we need to engineer software.

Major changes have taken place in the industry structure, the technology and the market since we developed the CADES system. Most of these emphasise the need for an Open system, in a Process sense, and in terms of flexibility of Tool interworking.

We are no longer concerned with the development of discrete software

components. Instead the major concern is with the “glueing” together of components to provide the IT component of some larger ecosystem. We need to recognise the endemic nature of IT, to recognise that our traditional software component is but one component of a much broader control system. We need to recognise the need for the support of mixed componentry, much of it never to be specified in terms to which our software development methods can sensibly relate.

We need to recognise the need for an ever increasing variety of methods, languages and toolsets, to recognise the need for support for system development “in the large” and for component re-use of a wide variety of component types. In particular we view the need for re-use of process model fragments as being of paramount importance.

7 Benefits/Concerns

The generic approach taken by the IPSE 2.5 project clearly implies a process of specialisation by many different groups in an organisation. Our approach to this has been to envisage the creation of an IPSE service function to help the various groups to create their PSE specialisations. In examining this service role it is particularly important to understand the wider aspects of the new technology's impact.

7.1 Benefits

Quality

A widely accepted definition of quality is that it is “conformance to requirement”. Such conformance is at risk unless we establish and keep to a process for stating and achieving the requirement. PSE's should therefore produce major improvements since they:

- improve our ability to establish appropriate processes, even in a changing environment, by reducing the cost of process change and greatly increasing the speed at which changes in process can be implemented
- make non-conformance to process difficult or impossible.

Predictability

PSE's will allow the automatic collection of a great deal of statistical information about active processes. This will mean, for example, a big improvement in our software metrics and a greatly enhanced ability to estimate time and costs for future developments.

A PSE can give the same planning capability as existing tools in terms of scheduling. They add, however, a capability for automatic update of progress since the PSE itself knows as soon as a task is completed. Managers can therefore replan rapidly to reflect the current situation without reliance on people to update the plan.

Management's knowledge that the process is being conformed to will give

them much greater confidence in the predictions of the plan. PSE's let people know what people ARE doing rather than what they SAY they are doing.

Rapid Execution of Process

The PSE is a means of automatic communication between people involved in a process. As soon as it is possible for a task to be performed, the person responsible for it can be advised through their workstation and in many cases all they require to do the work can be made available to them (e.g. documents and tools). Although, strictly speaking, this gives no change in productivity, processes could be completed faster. This could give competitive advantages, for instance, through a PSE supporting a complex bidding process or the process of getting new products to market.

Productivity

Every time work is done in an organisation there is the possibility that the experience could be used to make future work more productive. This knowledge asset has a nasty habit of evaporating as we constantly re-invent the wheel. PSE's can make a massive contribution to productivity by providing an environment in which it is easier to capture and re-use this knowledge in the forms of:

- standard processes
- product standards
- standard product components

Improved productivity will also be achieved in the following ways:

- better control will result in fewer mistakes being made and this will reduce the number of iterations
- less time will be spent looking for the standards and procedures which apply to an activity
- good, and bad, performance will be apparent much sooner, giving an improved basis for productivity-directed motivation.

Quick Reaction to Changing Business Environment

This is possibly the most important benefit that can be expected and one that could give an important competitive advantage. It is enabled by a greatly reduced cost of process change and a great increase in the speed with which process change can be implemented. PSE's will bring about a change in attitudes about how fast business processes can be changed and a readiness to seize the new opportunities.

7.2 Concerns

Managing Re-use of Process Building Blocks

Although PSE's create the right environment in which to re-use the process asset, we still have a great deal to learn about how this can be done. It fits in well with CIS's role to build process libraries and offer them as a managed service to help people to build their PSE's. It is less clear what should be in

them and how integrity can be maintained in the build process, though a number of approaches are being considered.

There are also management problems surrounding ownership, issuing authorities and compliance status which we suspect will require a much more formal approach to standards management than has ever been previously attempted. Some initial analysis of this has been carried out.

Requirement to Really Know the Process

It has been made abundantly clear by our work so far in constructing a SENSE IPSE that if processes are to execute on a PSE, they have to be formalised and recorded in much greater detail than has been the norm. This will not be considered a bad thing by many but it does have to be recognised. Once we have a good arsenal of tools and methodology for capturing process this will be less of a problem.

It should also be recognised that what we do to manage process through manuals is often to exaggerate discipline in order to combat anarchy. Thus it is common to insist that no work on a stage should be started until the previous stage is signed off as complete. Sometimes we mean this, but often we rely on people to break the rules when it is obvious that they should (while telling them not to). The PSE demands that we capture the true process which, in this case, could in certain circumstances allow work to continue at commercial risk.

7.3 Conclusion

The impact of PSE's on organisation is likely to be considerable, even dramatic. We have started to assess four kinds of impact:

- on the individual
- on groups of co-operative workers
- on the role of management
- on society

Individuals may view the coming of PSE's as a benefit through a reduction in the frustration caused by mistakes and through satisfaction at their own improved productivity. A common reaction, however, will be that the increased control is unwelcome and that the advent of "big brother" is suspected. This can be countered by managing the degree of constraints that are inherited by each person in the process-defining hierarchy. But managing them in a true evolutionary way, since each execution of a process will usually cause a change in that process.

Our expectations of PSE's will be completely unfulfilled unless groups find themselves able to co-operate more easily. Whether this turns out to be like a dehumanised cog in a machine or a willing member of an orchestra is probably a choice over which management has some power.

It is not unusual to hear of seasoned professionals who feel that knowledge of process is like knowledge of an instrument. Once one has mastered the rules, one may excel by breaking them at the right moment. This could be true but probably belies the situation which most often confronts us which is more cacophony than symphony. It is fun to play in an orchestra because everyone can be relied upon to keep to the rules. Even jazz improvisations would be unpleasant without some minimal order.

To us, all of this confirms the need for a process-based rather than an entity based core for a PSE.

References

- 1 PEARSON, D.: "CADES" Computer Weekly, July 26th, August 2nd, August 9th, 1973.
- 2 WARBOYS, B.C. and PRATTEN, G.D.: "CADES – Principles" Seminar, Oxford University, February 4th, 1976.
- 3 WARBOYS, B.C.: "VME/B a model for the realisation of a total system concept". ICL Technical Journal, November 1980.
- 4 BUXTON, J.N. and RANDELL, B.: "Software Engineering Techniques", Report on NATO Science Conference, October 1969.
- 5 NAUR, P. and RANDELL, B.: "Software Engineering", Report on NATO Science Conference, 1968.
- 6 SNOWDON, R.A.: "IPSE 2.5 Technical Strategy", IPSE 2.5 project document 060-00131-2.2.
- 7 JONES, C.B. and MOORE, R.: "An experimental user interface for a Theorem Proving Assistant", IPSE 2.5 document SE13/29/234.
- 8 DIJKSTRA, E.W.: "The Humble Programmer", CACM NO 10, Vol 15, 1972.
- 9 KOESTLER, A.: "The Act of Creation", Macmillan, 1964.
- 10 ATKINSON, M.P., MORRISON, R. and PRATTEN, G.D.: "A Persistent Information Space Architecture", proc. 9th Australian Computer Science Conf. 1986.
- 11 VEASEY, P.W. and POLLARD, S.J.: "Preparing the organisation for IPSE". ICL Technical Journal, November 1986.

An Introduction to the IPSE 2.5 Project

R. A. Snowdon

STC Technology Ltd., Newcastle-under-Lyme, Staffs. ST5 1EZ, UK

Abstract

This paper gives a brief overview of the Alvey IPSE 2.5 project. This is a relatively large project investigating certain features of advanced environments supporting systems development activities. The paper concentrates particularly on the idea of *process modelling* whereby support can be given for the processes by which development activities are carried out.

1 Introduction

The IPSE 2.5 project is concerned with the problem of how computer systems can be used in the development of software based information systems.

The project is being carried out under the UK Alvey Programme Software Engineering Strategy¹ by a consortium comprising STC Technology Limited, International Computers Limited, University of Manchester, Dowty Defence and Air Systems Limited, SERC Rutherford Appleton Laboratories, Plessey Research Roke Manor Ltd. and British Gas plc. It is of 4 years' duration, finishing at the end of September 1989.

The name "IPSE 2.5" is due directly to the identification in the Alvey programme of three generations of the so called Integrated Project Support Environment – IPSE 1, IPSE 2, IPSE 3. The IPSE 2.5 project lies somewhere between the second generation, characterised by the use of databases, and the third generation, characterised by the use of artificial intelligence techniques.

The acronym IPSE (Integrated Project Support Environment) has come to describe a computer based system providing a variety of facilities working together to support people who develop software based information systems. It is derived from the similar "Ada Programming Support Environment" (APSE) describing an integrated set of tools for someone developing programs in Ada.

The problem area of concern to the IPSE 2.5 project is that of computer systems which provide a set of facilities to support people and organisations (as opposed simply to individuals) concerned with all aspects of computer

systems development. In fact the project is both more limited and more general than this.

It is limited in the sense that the scope of an IPSE is ill-defined; it is always possible to think of something else that could be provided in an IPSE. The IPSE 2.5 project therefore has some very specific things with which it is concerned.

It is more general in the sense that it is quite clear that not only is there a vast number of different computer systems, but there are also a great many ways of developing them. Systems constructed to support the development of computer systems must therefore be, to some extent, general purpose.

The two major aspects addressed by the project may be characterised by the phrases "process modelling" and "formal methods". Process modelling is concerned with the need to provide support for the many processes involved in the production of computer systems. Formal methods are mathematically based approaches to software development. One of the original aims of the IPSE 2.5 project was to investigate how support for such methods could be integrated within a system providing support for the broader set of processes concerned with computer systems production. This paper gives an introduction to the project as a whole, placing somewhat more emphasis on the notion of "process modelling" than on those concerned with "formal methods". Readers interested in the latter are referred to⁵ for more details. It is planned that a later issue of the *ICL Technical Journal* will contain a paper which gives more details of the support for process modelling as developed by the project. However this edition does include a paper⁷ concerned with the application of the ideas within a company such as STC.

2 Motivations for a support system

At the risk of over simplification, the behaviour of most companies can be described in terms of the model shown in Fig. 1.

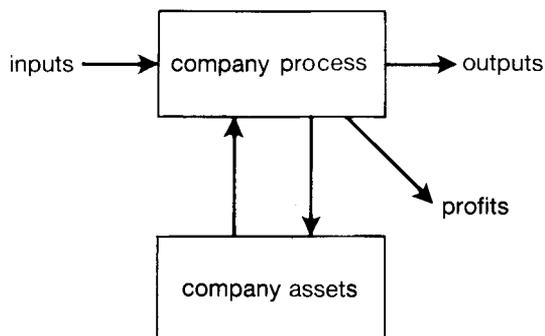


Fig. 1 Company process

A company uses its 'assets' (financial, intellectual, material) to add value to its 'inputs' in order to produce 'outputs' from which it can directly, or indirectly, increase its assets and make profits. This we may call the company 'process'.

For any given company in any given business it is clear that it must (generally) continue to improve its ability to carry out its business. This might mean that the cost of doing so must be reduced or that the quality of its outputs must be improved or both. The company process must become more effective. A critical aspect of this is the management of change.

Over the years many basic tools have been produced to support the production of computer systems. These include, for example, high level programming languages and compilers, testing systems, tools for building systems from components, management tools and a wide variety of others. However, tools such as these are only part of the answer. It is clear that simply having suites of tools which support specific activities is not sufficient, particularly in the context of the larger and more complex systems that are now being developed.

Thus the idea of a computer based system aimed at supporting the company process rather than the use of computers as individual tools within the process is seen as an important concept. This "process support system" is not a new concept; the basic notion of a business support system is a well established concept in several areas of computer applications. The application of these ideas to the complex area of information systems development is, however, more unusual. The gains to be made in corporate effectiveness, over and above gains to be made by simple technical advancement are, however, very important.

Whilst a process support system should be of great benefit to a company producing information systems, it is still the case that the individual tools and techniques by which computer systems are developed are at present primitive. The reliability and quality of computer based systems produced by the application of these techniques still leaves much to be desired. A support system which simply allows companies to apply such techniques more effectively does not directly address these basic problems. As new techniques emerge it is necessary for them to be properly integrated into the support system concept.

3 What is an IPSE?

An IPSE is a computer based system supporting those parts of the overall business process which are concerned with the technical development and maintenance of an information system. This process might start at the requirements definition phase, continue through the stage when a product is completed ready for shipment to a customer and conclude with activities concerned with supporting the product in use. An IPSE might be concerned with supporting technical activities outside this scope – the exact delineation

need not be well defined. Generally, however, an IPSE will not be concerned with the business or commercial aspects of the overall company process. This does not mean that the management and control aspects of the production process are outside of the scope of an IPSE. On the contrary, these aspects are very much an IPSE concern! Note that the exclusion of business and commercial aspects is, essentially, an artificial division and derives from historical reasons as much as anything else.

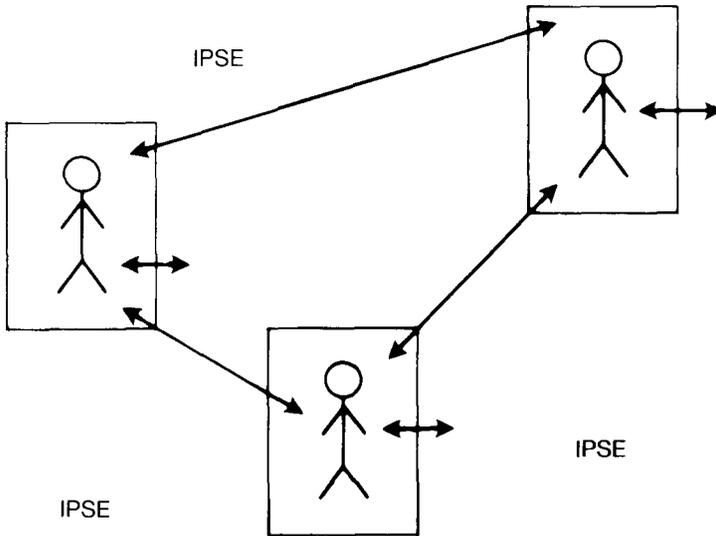


Fig. 2 Working environments

As in Fig. 2, the IPSE provides the environment within which individuals work. As far as each individual is concerned, the IPSE provides him/her with an appropriate working context. Actions carried out by the individuals working within an IPSE will cause changes to those working contexts. Of course, changes may also be brought about by actions initiated from other sources, either external to the IPSE or from within it (e.g. tools). The appropriateness of working environments to the particular tasks to be undertaken will have significant impact on the effectiveness of working within such an IPSE.

Of course people play the most important roles in developing information systems. An IPSE must therefore be seen as providing support for human beings as they carry out their activities within the overall development process.

Particular IPSEs (i.e. particular computer based systems) will vary in the facilities provided as well as the scope of the overall process which is addressed. It may be the case that a specific IPSE provides support for systems to be developed in only one programming language or for one type

of computer. Support may be concentrated on a particular method of systems design or for small or large projects. The degree of "integration" will vary from an IPSE formed from a loosely coupled set of tools to one where the facilities provided are tightly and coherently bound and presented to the users.

In general an IPSE provides a means of storing components and descriptions of the system being developed and a set of tools which can be used to create such components and descriptions, and ultimately the system being developed.

Most existing IPSEs are either almost completely concerned with providing support for the technical functions within the development process or provide relatively separate support for technical and managerial activities. Such IPSEs are based on a "store plus tools" model which emphasises the product being produced rather than the *process* which is being followed. This has the effect that such IPSEs are not able to provide appropriately distinguished working environments as there is little or no information available about the state of the process in order to be able to do so.

4 What is IPSE 2.5?

Specifically IPSE 2.5 is a project whose objective is to develop and demonstrate a small number of IPSEs each constructed on the basis of particular characteristics. The name IPSE 2.5 is generally used to describe such systems.

The characteristics which are of principal interest to the project are concerned with advancing beyond current IPSE projects in two important areas. These areas have been alluded to above and are specifically:

- 1 to raise the level of integration within the support system above the store-plus-tools approach by developing IPSEs which have knowledge of the processes by which information systems are developed. The term used to describe this idea is Process Modelling.
- 2 within this more integrated framework, to provide effective support for the use of the emerging development methods based on mathematical formalisms. These methods, known as formal methods, offer very significant advantages to systems developers over traditional approaches. However, it is clear that good support is needed if these approaches are to be widely available to industry.

An IPSE 2.5 system will provide a means of holding information about products and projects, it will support configuration management and version control, it will provide means by which existing tools can be used and it will be equipped with state of the art MMI facilities to ensure the achievement of effective man-machine synergy. One of the major challenges of the project is to provide all of this integrated with Process Modelling and support for Formal Methods in a clear and cohesive manner.

The IPSE 2.5 project is, of course, one which is investigating a number of advanced concepts. The IPSEs being produced by the project have an experimental nature, the properties of which will be evaluated as an important part of the project itself.

5 What is Process Modelling?

In section 2 the idea was introduced that an IPSE is about supporting the processes by which information systems are developed and thereby making these processes more effective. The term Process Modelling is used to describe the production of models of such processes and the use of these models in an IPSE. Though these basic ideas are not new, the application in the IPSE 2.5 project has some novel aspects.

The idea may be illustrated by analogy, by considering, for a moment, not a support system for the development of information systems but an automated manufacturing plant, maybe a production line for motor cars.

Such a plant might comprise a number of machines, each capable of carrying out a particular operation on objects passing through the plant. The successful operation of the whole plant requires the proper synchronisation and control of all the machines and of the flow of objects between them. This is the responsibility of a "controller", provided in the plant by a computer system. We assume that raw materials are made available to the start of the process (in fact to the first machine in the line), operated upon in the appropriate manner and order by the individual machines in the plant, with the final manufactured objects emerging at the end. The "controller" system must be provided with some sort of overall model of the plant and the required behaviour so that it can cause the individual machines to operate at the right time. This model might be "hardwired" into a special purpose controller which can only control very specific processes, or it may be loaded into a more general control machine much as a computer program is loaded into a general purpose computer. As the control machine "obeys" the model, so the appropriate process is enacted by the real machines in the plant.

There may be some conceptual distance between an automated manufacturing plant ("populated" by machines) and a company producing information systems ("populated" largely by human beings), but the analogy should be clear. The "controller" in the manufacturing plant is part of the system supporting the process. Thus it might be expected that there be something similar in an IPSE supporting the process of system development. The Process Model to be loaded into an IPSE would hardly be as straightforward as that for a manufacturing plant. It is one thing to describe the sequence of operations for a machine, it is quite a different (and generally unacceptable) matter to expect to be able to prescribe in detail the behaviour of a person or persons carrying out the intellectually demanding tasks of information systems development. It is however, quite reasonable to consider a style of model which is, perhaps, descriptive of the *goal* of a task as opposed

to the imperative form that might be characteristic of an automated manufacturing plant.

Of course, the factor of "change" is one which is of much greater significance in the process of producing information systems. In an automated manufacturing plant, the process is relatively well-defined and may be left to run with little intervention (for such is the intention of automation). The process of producing an information system is much less well understood and requires a significant amount of monitoring and adjustment. These processes (of monitoring and adjustment) are essential parts of the overall process and must therefore be supported by the IPSE. Companies involved in producing information systems do, of course, recognize the importance of these management processes. The paper⁷ elsewhere in this issue of the *ICL Technical Journal* notes the relationship with the STC SENSE methodology for systems development.

6 Support and use of process models

A key component of an IPSE 2.5 support system is something which is analogous to the "controller" in the manufacturing system. This component (called the PCE – Process Control Engine) can be loaded with a Process Model of the activities to be carried out by the staff of the company or project using the IPSE. The IPSE 2.5 project is developing a language in which such models are expressed. This language is called the Process Modelling Language, or PML. Early work reporting on ideas contributing to this language can be found in⁶.

The Process Model could be of an imperative nature as in a manufacturing plant, but more generally it will be less stringent, describing the tasks that need to be carried out and the constraints that should be observed.

According to what is described in the Process Model the PCE provides appropriate working environments for each of the staff using the IPSE to carry out the tasks involved in developing a system. These tasks may be technical or managerial. In the former case the staff member will make use of tools as provided in his working environment to develop further the system being produced. These tools may perhaps be tools developed elsewhere (and incorporated into the environment through standard interfaces) or may be tools specifically developed by the IPSE 2.5 project and more tightly bound into the environment. Certain tools developed to enable the use of formal methods come into the second category. The environment will also provide a storage component (and configuration and version control facilities) in much the same way as current IPSEs do, except that access to the store will be controlled through a working environment and hence, ultimately, under control of the Process Model.

The working environment provided for managerial activities will generally provide additional features concerned with the monitoring and changing of Process Models.

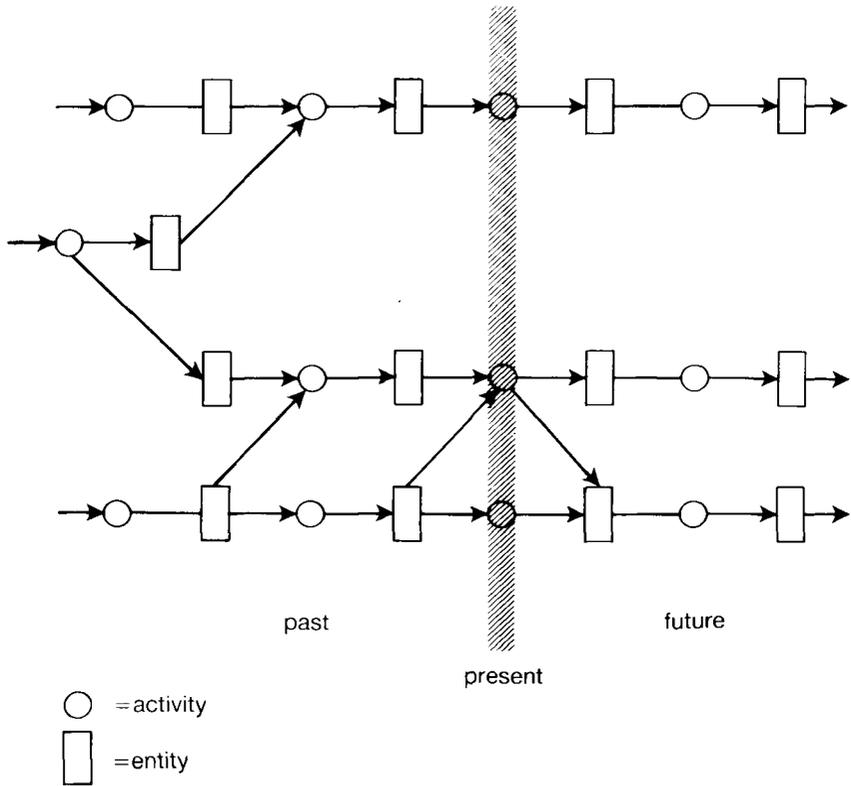


Fig. 3 Past, present and future

Figure 3 shows a view of a process model expressed as a network of activities, shown as \circ and entities, shown as \square . In this diagram, the current set of activities are indicated by their lying on a time line labelled "present". The model to the left of this line represents the past, whilst that to the right represents a prescription for what can happen in the future. This future is rather like a plan. By definition, this "plan" has been produced by an activity (usually described as a managerial activity) at sometime in the past. Such an activity is therefore amongst those to the left of the "present" line. Management activities are part of the process model. Thus a process model not only describes the technical activities to be carried out in the development of a system, it also describes the managerial process as well. Very importantly, it is necessary that it be possible to support activities concerned with monitoring and change, so that the process model can be altered as necessary to align better with real intentions. Reverting to the manufacturing plant analogy for a moment, it is as if there is a machine in the plant, under the overall control of the "controller" which monitors the plant and is able to "reprogram" the controller at appropriate points.

This particular area offers some challenging problems, particularly in terms of the implications of dynamic binding and the need to control the impact of change in an executing model.

A third class of user of an IPSE is the engineer who is responsible for ensuring that the IPSE itself operates properly. Appropriate working environments for tasks associated with such activities are established in the same manner via Process Models loaded and activated by the PCE.

7 Tools

The description so far has concentrated on the idea that people involved in the process of developing an information system are assisted by the provision of working environments appropriate to the tasks they have in hand. As tasks are carried out so the environment changes, both for a particular individual and for others affected by his or her work. No mention has been made so far about how tasks are carried out, in particular the provision and use of computer based tools. In a complex and challenging activity as are those involved with the development of information systems, it is absolutely necessary that use be made of computer tools to carry out certain tasks. As part of the working environment provided in order to carry out some activity, an IPSE must provide access to relevant tools and ensure that use of these tools is properly integrated with other relevant objects. Thus, if an activity is likely to require the application of some analysis tool to some object, then the IPSE should ensure not only that the analysis tool is available, but that it can be applied to the object. In some circumstances, the use of a particular tool may not require its selection by a person, but rather that the tool is applied automatically as a consequence of some other action. In any case, there is a clear requirement that computer based tools be considered as an integral part of the *process*. The implication is that the Process Modelling Language, and thus the PCE, must provide means by which tools can be incorporated into an IPSE. In fact, there are different types of tool which must be considered. The most important are the vast number of tools which already exist, having been built for particular purposes and, in all likelihood, for particular computers and operating systems. As far as these are concerned, means are provided, through PML and through tailorable constructs within the PCE, by which such tools can be added to and used in an IPSE 2.5 system. The OBJ and MALPAS tools referred to in the next section are examples. At the other extreme are tools which are really just a form of process model and are completely defined in PML. These tools are, of course, peculiar to the IPSE 2.5 culture and, whilst being easily integrated with the rest of a model, cannot be made use of in other situations.

8 Support for Formal Methods

The project is concerned with providing support for formal methods from two points of view.

- 1 the use of formal methods in an industrial setting
- 2 further research to extend the scope and applicability of formal methods, in particular through advanced tool support

If formal methods are to become more widely used in industry they must include elements which are necessary in such a setting. The methods emerging from the research laboratories must be 'industrialised'. This requires that formal methods development processes are produced which address project management concerns, attend to the problems of multi-member projects, relate to corporate quality procedures and so on.

The IPSE 2.5 project aims to construct an IPSE supporting such a formal methods based development process. In fact the project will develop an IPSE to support a combination of the OBJ (see ³) and MALPAS (see ²) techniques. This combination is used within the Plessey Company and hence is extremely relevant to the purposes of the project. The IPSE will be constructed using the Process Control Engine and Process Modelling Language referred to above. Commercially available tools for OBJ and MALPAS will be integrated within the model for use at the appropriate points within the development process. Thus, the development of such an IPSE does not rely on research into formal methods or on the development of tools of an advanced nature. What is important is that the major characteristics of a formal approach to development are well understood, captured in the total process and supported by the IPSE.

Such a formal methods IPSE will necessarily reflect the current rather limited level of support for formal methods. More advanced support is still a very active research topic. There is, in the IPSE 2.5 project, a strong research thread aimed at examining some of these issues, particularly those concerned with formal reasoning.

As development proceeds from specification to implementation, opportunities (or obligations) arise to make proofs about properties of the system being produced. The mathematical nature of the specifications makes this possible. Support is needed both to identify the proof obligations and to help the developer carry out the proofs themselves. Both the determination of obligation and the means of carrying out the proof depend to some extent upon the method and on the form of the underlying mathematical basis.

In this part of the project advances are sought in terms of general purpose tools both to determine proof obligations and to assist in carrying out the proofs. In the latter area the approach being followed is to develop an interactive proving system. Emphasis is placed on the interactive nature of such a tool with research being particularly directed at the advantages which might be gained from the use of a modern, powerful workstation with a relatively large screen and a mouse.

A related topic of research is concerned with symbolic execution and its

support. It is sometimes the case in formal methods that it is useful and appropriate to explore the properties of a system rather than make and prove hypotheses. Symbolic execution can be a powerful tool for such purposes.

9 Status

Work on advanced support for formal methods has been carried out separately from the remainder of the project. An early prototype interactive proof system was developed to explore both the use of Smalltalk for such an application and to explore appropriate styles of user interaction. The results from this system (named Muffin) are now being used in the development of a more realistic tool supporting a wider variety of logics and formal development methods such as VDM. ⁵ reports more fully on this work.

As reported in ⁶, the starting point for PML development was the language RML ⁴ and the general field of conceptual modelling. This led to an initial definition of PML and a prototype support system built, again, on top of Smalltalk. A number of experiments have been carried out with this system including models of processes taken from the industrial partners in the project. Developments are now underway to construct a system capable of supporting realistic numbers of people working on relatively large projects. This implementation will be the basis of demonstrations and evaluation exercises to be carried out during the final stage of the project.

Already, however, explorations have taken place to investigate the application of the results of the IPSE 2.5 project within STC. These include the capture of aspects of the CIS (Corporate Information Services) SENSE approach to development in a process model and the installation of that model on a PCE to give a prototype tool supporting the use of SENSE. Links are also being made with different tools in ICL Mainframes Division (particularly with CADES and system construction tools) to add process modelling capabilities as part of the VME Software Factory exercise. This has required that formal process models be constructed of relevant Mainframes development processes. It is hoped that reports of such work will appear in a later issue of the *ICL Technical Journal*.

References

- 1 Alvey Programme Software Engineering Strategy. November 1983.
- 2 BRAMSON, B. D.: Tools for the Specification, Design, Analysis and Verification of Software. RSRE Memorandum 4063, RSRE, Procurement Executive, Ministry of Defence, RSRE Malvern, Worcs., June 1987.
- 3 GOGUEN, J. and MESEGUER, J.: Rapid Prototyping in the OBJ Executable Specification Language. SIGSOFT Software Engineering Notes, 7(5): 75-84, December 1982.
- 4 GREENSPAN, Sol J.: Requirements Modelling: A Knowledge Representation Approach to Software Requirements Definition. Technical Report CSRG-155, Computer Systems Research Group, University of Toronto, March 1984.
- 5 JONES, C. B. and LINDSAY, P. A.: A Support System for Formal Reasoning: Requirements and Status. In VDM'88, VDM-Europe Symposium 1988, Dublin, Springer-Verlag 1988.

- 6 OULD, M. A. and ROBERTS, C.: Defining Formal Models of the Software Development Process. In Pearl Brereton, editor, Software Engineering Environments, Ellis Horwood 1987.
- 7 WARBOYS, B. C. and VEASEY, P.: Twenty Years with Support Environments. ICL Tech. J., 1989 this volume.

The case for Case

Fred Russell

ICL System Strategy Centre, Bracknell, Berkshire

Abstract

At least half the world's development staff are busy maintaining systems – at least half this effort is non productive. One quarter of the remainder are building systems that will never be used. The remainder are working at about half the efficiency called for in the plan. Why is this so? This paper examines some of the reasons and the extent to which computer assisted software engineering (case) tools may solve part of the problem.

Introduction

This paper very briefly examines some of the problems generated by today's software technology and the demands placed on it by the ever increasing complexity and sophistication of applications. It explores the potential of Computer aided software engineering (Case) technology for helping to solve these problems with some idea of its application and benefits.

I imagine almost everyone interested enough in software development to be reading this article will have little difficulty in finding an echo of their own experiences in the above statement. If you break the statement down you will see that around 75% of the world's software effort is being lost due to excessive maintenance, poor specifications, cancelled products, massive overruns and inefficient practices. Some more 'facts' for you to ponder:

- 1 US Army Study of projects totalling about \$7M. Showed:
 - 47% delivered but never used
 - 29% paid for but never delivered
 - 19% abandoned or reworked
 - 3% used after change
 - 2% used as delivered.
- 2 Maintenance accounts for 67% of life-cycle cost.
- 3 Depending on the survey source you use, between 25% to 75% of all software projects started are cancelled, often because projects are deemed to be obsolete before completion.
- 4 About 1% of large systems (50K + lines of code) are finished on time, within budget and meet user needs. The average large system is a year late and costs almost twice the original estimate.

(Anyone interested enough can find many more such examples in recent issues of Software Engineering Notes (ACM) in Peter Neumann's series on "Risks to the public in computer and related systems".)

If these sort of problems are not bad enough, they are exacerbated by demand for software products growing by about 12% annually whereas productivity is rising by only 4%, both figures compound. This means that demand for output is outstripping potential supply by 3:1, and growing.

One of the key reasons why we have these problems of heavy maintenance, non-delivery and delays is the high rate of defects and bugs in software. A survey by Software Productivity Research, Inc. revealed that an average large program will have 50/60 bugs per 1000 lines (serious bugs, that is).

As a final point, there is a people problem. Productivity aside, the demand for people far outstrips the potential supply by a factor of at least two. In absolute terms the figures for numbers of people needed defy belief, like a million for defense related projects in the States by 1990. The question is not so much where all these people are coming from, but where are they going to?

Developers cannot continue to indefinitely expand their office spaces to accommodate this growing army of (largely non-productive) people. Nor can they or their user population continue to accept the ever increasing backlog. In any case there is an old principle of Economics called diminishing marginal utility at work here; if you just keep on throwing more people at a problem you invariably reach a point where they get in each others way. Also, it is not just a problem of the right skills match of the people, the intangible nature of software and its inherent flexibility creates a problem in itself.

The trouble is that you cannot throttle down this ever growing demand either. In a sense, the software industry is guilty of that most cardinal of all business sins, overtrading. We have become the victims of our own myth in the expectations we have created in the market for our products. As the power of the hardware expands and becomes cheaper and more flexible, the range of potential applications remorselessly increases, fueled by growing user awareness of what may be possible.

If the demand exceeds the human resources available to meet it and these extra resources are unlikely to be forthcoming in any reasonable timeframe then higher productivity has to be the answer.

Software engineering context

It is the theme of this paper that productivity is a direct function of the quality of the tooling used to support development, and that Case is an essential component of any solution.

Just in case I get accused of doing some overtrading of my own, let me quickly establish the true context of this paper. Software engineering is the background theme. As most of you will know, its not a simple thing to define, so perhaps if we identify the main areas it is concerned with as shown in Table 1.

- | | |
|--------|---|
| 1..... | improving the softwear development process itself |
| 2..... | methodology for life-cycle control |
| 3..... | softwear metrics |
| 4..... | project management techniques |
| 5..... | formal methods of specification |
| 6..... | structural methodologies |
| 7..... | automation and support |
| 8..... | re-use of software components |
| 9..... | verification, validation and testing |

Table 1 Main areas of softwear engineering

Areas 5, 6 and 7, are the areas I want to explore with special reference to Case tools.

The term "software engineering" was coined as long ago as 1968 at a NATO Science Committee conference set up to examine the (then) problems of developing software. Although it may not seem like it to some of us, quite a bit of progress has been made since then, masked to some extent by the increasingly complex products the industry has been expected to produce. Some of these areas of improvement are: better methods of project control and quality management; development of the life-cycle model; the evolution of formal and structured methods and, most interestingly, the development of automated tooling to support these techniques.

One of the important lessons to emerge is that all these techniques fit together, they are not truly independent processes. To use them in an effective manner therefore requires an integrated approach which itself has significant impact on the organisational culture in which these tools are used. The concept of integrated development supported by software tools plus some kind of automated infrastructure leads to the idea of the integrated project support environment (IPSE) in the development of which ICL is a major player through the Alvey and ESPRIT programmes.

The diagram in Fig. 1, from the DTI/NCC STARTS Guide, second edition, illustrates the interdependence of these facilities

In order to be effective the methodological structure illustrated above needs to be reinforced by or allied to a strong conceptual model of the design process. The life-cycle model provides one such conceptual basis for organising and managing the production of software, and we see this embedded in ICL's phase Review mechanism. There are many variants of this linear model, ranging from Barry Boehm's Waterfall approach to the STARTS

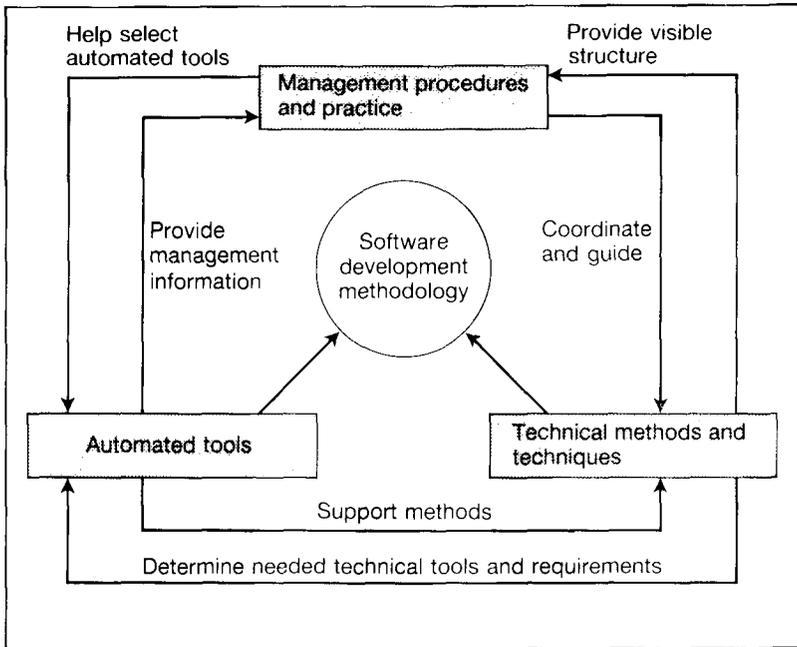


Fig. 1 Software development methodological framework

model illustrated in Fig. 2, the Prototyping advocates tend to disagree with the linear model as a foundation methodology for a number of very sound reasons, but if you examine any Prototyping model, embedded within it is a linear one; what differs may be the degree of formality assigned to the phases.

Effectively, no matter how you play around with diagrams, Prototyping does not basically invalidate the linear model since Prototyping iterates up and down the left hand side of the 'V' until a match to user requirements is achieved. Then it may form part of a formal process, either starting all over again, this time with the prototype serving as the requirement, or it may form the product itself, migrating up the right hand side of the 'V'. Depending on what type of Prototyping techniques employed, e.g. exploratory, experimental or incremental etc. Boehm's Spiral model is probably the closest to an integrated linear/Prototyping approach.

The coding process itself occupies an ambiguous position in the linear model (Prototyping aside). In fact coding often begins in a real world situation before design is complete, it continues through the testing and debugging phase on to the maintenance phase which, after all is what you would expect since the code *is* the software component of the system. It is a useful construct to conceive of coding as occurring at a single point in the process, but reality has its own way of dealing with abstractions.

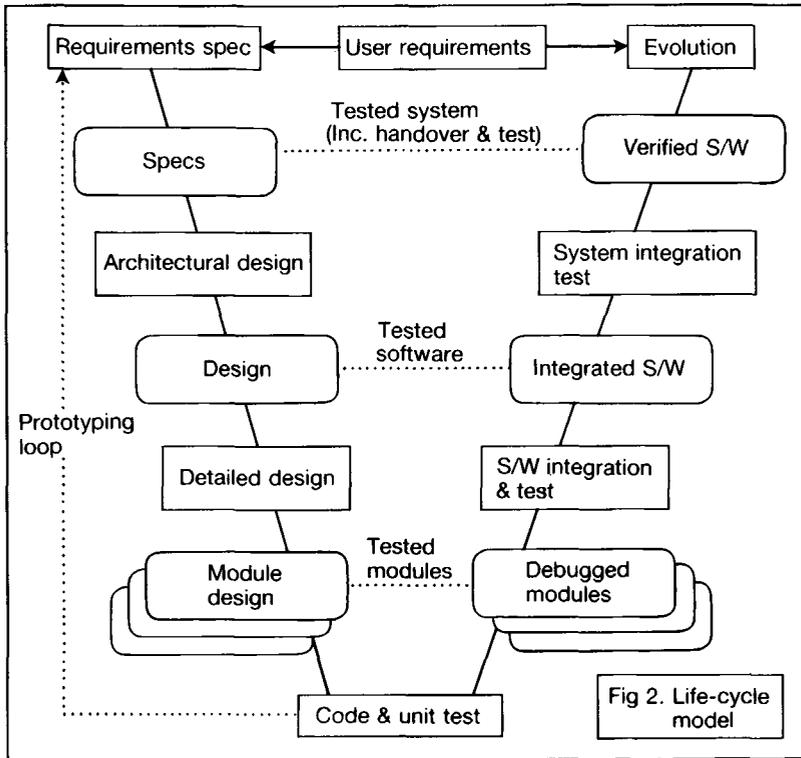


Fig. 2 Life-cycle model

Requirements, however, seem central to the whole problem. While it is a sound principle to define the quality of a product by the degree to which it conforms to requirements, how do we test the quality of the requirements themselves? The requirements specification can be an uneasy compromise between end-user needs and the developers technical or marketing constraints, a kind of balancing point between marketing pull and technology push. This 'front-end' phase is probably the most difficult to get right and has been recognised in software engineering by the plethora of formal and structured techniques developed to give some coherence to the process. It is interesting that it is this area that Case really began to emerge as a way forward with its many variants of analysts workbenches.

In the above brief dissertation four issues seem to emerge as the foundations for Case development. These are:

- 1 The need to construct integrated development environments – including project management
- 2 the coding problem, its correctness and the extent to which it drifts from the original requirement

- 3 the need for an adequate set of requirements couched in a form which reduces errors in both logical structure and relevance to need
- 4 the need for Prototyping capability as an alternative where requirements are uncertain as they certainly will be where innovative or speculative development is needed.

Which, I think, brings us to Case itself, what it is and where it is going.

Case defined

As an acronym, Case is usually taken to mean “computer assisted software engineering”. Some definition replace ‘software’ with ‘system’ to indicate the potential range of these tools, but for the purpose of this paper we will stick to software as the primary area. The term “Case” was reportedly coined by John Manley in 1981 who was then the head of the Software Engineering Institute at Carnegie Mellon University.

The significance of the term ‘Case’ lies in the recognition that software could be used to assist in the development of software and may have its ideological roots in CAD. Case is a US term only just beginning to become common currency in the UK. On the other hand, ‘Ipse’ (which is a British extension to the US ‘Apse’ concept) is widely used over here to describe a class of tools supporting project management activities and tool integration. The Iperse seems to have no precise equivalent in the US where Case development tends to concentrate on the technical aspects of software development, such as analysis, design and coding. In Britain we concentrate on project management and the construction of ‘super’ operating systems for software developers, providing public tools interfaces for attachment of technically based tools: Iperse 2.5, PCTE and ECLIPSE are instances of this type.

Generally, Case tools tend to cluster round three areas of software development. These are shown in Table 2.

Major areas	Case clusters
Analysis and design	Analysis and design aids
Coding	Coding generators and programming support environments
project control	Integrated project support environments

Table 2. Clustering of Case tools

The problem that many people have with Case is knowing its limits and where it fits into the spectrum of software products. The Ovum report places it as shown in Table 3. Below.

Application packages	Vertical Horizontal	Banking, Retail etc. Order processing, accounting, OA, <i>project management</i> etc.
Development tools	Re-usable libraries	System <i>kernel</i> s, <i>math</i> . <i>subroutines</i> , <i>I/O routine</i> etc.
	Case	<i>Analysis and design aids</i> <i>Code generators,</i> <i>programming support</i> <i>environments, Ipse's</i>
	Application tools	4GLs, DBMSs, report generators, query languages, spreadsheets, graphics, sorts etc.
System software utilities		Operating systems, communications, <i>compilers</i> , <i>cross-compilers</i> , <i>linkers</i> , <i>debuggers</i> , <i>editors</i> etc.

Table 3. Case position in software product spectrum
Italics indicate areas where Case is relevant

This classification positions Case in a fairly central position in the range. The narrowest definition of Case is that they are only concerned with analysis and design processes.

The role of Case

The real objective of Case is to automate and support all the functions of the development team that are needed to develop and maintain software. As you would expect, however, the functionality actually provided by Case is a compromise between what its potential users would really like and what functions are most appropriate for automation. So, Case tools tend to focus on the more routine and prescriptive level tasks which none of us enjoy doing anyway, freeing up creative energy to concentrate on the higher level decisions and activities. For instance, in its most simple form a Case tool will provide graphic drawing and editing facilities to replace the pencil, paper and templates we are used to, with the added bonus that the diagrams are automatically verified for consistency, correctness and loose ends. **It is in removing the drudgery from development that Case contributes most to productivity and quality.**

One of the key effects of Case however is its impact on the way a user organisation needs to orient itself round the use of these tools. Some user organisations in the US have found, for instance, that their skills requirements are shifting away from the designer/programmer team towards the analyst/designer team where the skills orientation is closer to the application area.

Case cannot be effectively used in isolation. It must be used in conjunction with a very specific life-cycle strategy and the methods and disciplines needed to support that particular view of the life-cycle. Bearing in mind that there are literally hundreds of methods currently in use this poses significant problems for both the user and the developers of Case. Should Case be completely method independent, should it support the methods in use or should it enforce them?

It is worth a quick look at the three areas of software engineering impacted by Case (see Table 1). These are:

- Life-cycle models
- Structured methodologies
- Formal specifications

Life-cycle models

There is no such thing as a definitive model, each organisation having its own variant, some having none at all. The underlying problem with life-cycle models of the type shown in Fig. 2 is their relative inflexibility. A large product can take up to two years to develop and by that time the original requirements may have become invalid. This problem tends to push the updating of the original requirements into the maintenance phase which, according to Boehm's COCOMO metrics also pushes up the cost by a factor of fifty. Nevertheless, the life-cycle model is in almost universal use and its use has been strongly influenced by defence contracts requirements.

An instance of this is the US DOD standard 2167 where the procedural and documentation demands of the standard are almost impossible to meet without Case support. We have similar standards in Britain. It is worth noting here that, whatever the type of life-cycle model in use, there are probably six other underlying activities which continue throughout the project time frame. These are: project control, quality control, verification-validation and testing (VV&T), object management, Prototyping and documentation. Table 4 below illustrates where Case may be most relevant to a life-cycle or phased model.

Prototyping is a special instance of life-cycle models which attempts to avoid the relative inflexibility of the linear approach and, in fact, you would find very few Case tools that do not provide some form of prototyping. There is no doubt that regardless of the type of prototyping technique used the main advantages are the same – getting commitment from the user and feedback of evolving requirements. In many ways it is a better way of building large systems with its “build a little, test a little” approach. It can however provide significant administrative and project control problems for the unwary.

Structured methodologies

Structured methods have been around since the 1970s for coding; these were

			
Analysis and design aids			
Code generators			
lpses			
			
Requirements analysis.			
Design			
Coding			
Intergration and testing			
Maintenance			
Prototyping			
Documentation			
V V & T			
Project control			
Object management			
Quality control			

Table 4. Relevance of Case to life-cycle phases

extended to include design and then analysis. Their use is still very spotty and there is a tendency to regard their use as suitable only for very large systems or defence projects. One of the usability problems is the immense amount of documentation generated.

Structured methods consist of sets of rules and procedures for structural decomposition, usually based on diagramming techniques, generating data flow diagrams, entity models etc. Some of the techniques are underpinned by mathematical theories, for instance, Jackson's structured methods use proofs developed by Boehm and Jacopini in 1966 which state that any control structure can be expressed by using three component types – sequence, iteration and selection.

Some of the leading techniques are Yourdon and Information Engineering in the States, Jackson and SSADM in the UK.

All these techniques are naturals for Case and, as with prototyping, there are few analysis and design tools which do not use one variant or another. One of the really big advantages of Case in this area is in the ability to control the documentation problem.

Formal specifications

Although a good deal of effort is going into the development of natural language interfaces for computers, there is the problem that the semantic ambiguity of natural language, when used as a requirements language,

generates uncertainty and interpretation conflicts between the user and the designers perceptions of what is really wanted.

Formal specification methods are (slowly) emerging as a possible way to avoid ambiguity of meaning. They have, however, significant usability problems in that they require considerable skills in set theory and other branches of mathematics that are not commonly available. Equally, there is a problem of where to stop. There is a natural tendency to merge the requirements and design phases as a virtually single process. Its use is not very widespread, particularly in information system design, however, as an instance of its use, ICL uses "Me too", which is a subset of VDM to specify parts of its "Problem and resolution information system" (PARIS) a system which will provide ICL with information about system problems with its products.

Some leading examples of these techniques are VDM (Vienna Development Method) by IBM, Gist by Information Science Institute in California, Gipsy by Computational Logic and Z by Oxford University. Clearly these techniques will benefit by Case support but there are not too many instances at the moment.

What do these tools actually do?

The clusters defined in Table 2 are based on the particular strengths of tools categorised under these headings. In fact many of the marketed versions contain some aspect of all three cluster characteristics. For the purpose of illustrating what these tools actually *do* then the cluster approach is the best way to describe them, so lets have a look at these categories.

Analysis and design aids

Typically, these are PC or workstation based and are designed to help the analyst create and edit diagrams, usually of the structured methodology type such as data flow diagrams, entity models and life histories etc. Tools in this cluster typically provide for several styles of diagrams and some provide screen painters. Diagrams are frequently provided for both data and process modelling. Sometimes, as with the Information Engineering Facility from Texas Instruments, there is separation between diagrams used for analysis and those used for design.

Generally, these are low cost systems, it is easy to see what they do and are usually easy to learn to use. Most importantly, for many intending users, they can be used as 'point' tools in that they do not necessarily demand major overhauls of the user organisation's development methods.

One of the market leaders in this field is Exceleator from Index Technology. As a drawing device these systems have many obvious advantages over pencil and paper:

- quicker
- errors easily put right
- results are visually consistent
- diagrams can grow in any direction
- diagram components can have attributes and properties associated with them
- design rules, syntax correctness and completion criteria can be automated enforce the standards of the chosen methodology

However, today there is no room in the market for tools that simply draw pictures. Just about all the tools in this class provide data modelling backed up by some form of data dictionary system. Many of these provide for export of their design data to (say) a mainframe data dictionary system or can be passed on to a code generator. Increasingly, these products are expanding away from their 'point' bias to become toolsets within some project management environment, or are part of a network in which object management is the key requirement. The emphasis is now moving away from tools based on design data towards design *knowledge*. Knowledge-based Case tools are beginning to appear such as the Bachman/series and Exsys. My own project, part funded by the ESPRIT HUFIT programme, is producing a knowledge based Case tool called INTUIT.

Some instances of these tools currently on the market are given in Table 5.

Core Workstation	British Aerospace
Kalix/Kindra	British Telecom
Automate Plus	Cullinet/LBMS
Developer Workstation	DBMS Inc
Quikbuild Workbench	ICL
Excelerator	Index Technology
Information Engineering Workbench	Knowledgewear/Arthur Young
MacCad	Logica SDS
Speedbuilder	Michael Jackson Systems
Design Aid	Nastec
SQL Design Dictionary	Oracle Corp
Information Engineering Facility	Texas Instruments
Analyst/Designer Toolkit	Yourdon Inc

Table 5. Analysis and design aids

Code generators

This range of products are those which output third generation languages such as Cobol, PL/1, Ada and Lisp. Usually there will be a range of support software such as compilers, linkers, interpreters etc. so that they can generate executable code. The natural consequence of having this facility means that all this class are capable of prototyping, for instance by animation of screen painted sequences. It often does not matter that the language in which the prototype is generated is not in the eventual target language since, depending

on whether the prototype is intended to become the product, as with incremental development, the purpose will have been achieved of defining in a pseudo run-time mode the intended functionality of the proposed system.

There is obviously some similarity with 4GLs in this respect, but there tends to be advantages with using the high level code of generators because they can use less computer resources than 4GLs. Even so, some Case tools are integrated with 4GLs.

It is in this type of Case tool that we see re-use put to work. They usually operate by bolting together bits of code forming a template in which the gaps are filled with variables or labels derived from the screens or forms the designer has previously painted in the design phase. Where there is extensive computation involved then extra code may be required or perhaps could be input via some kind of spreadsheet metaphor. There is little doubt that this aspect of Case will benefit from knowledge based support guiding the development of the coding process. The Knowledge-based programmer assistant which was part of the original PCTE programme used a form of conditional logic to support the user.

Productivity increases of some magnitude tend to flow from using this form of Case. The Ovum report lists the following as some of the reasons for this:

<i>Activity type</i>	<i>Example</i>
- Specification in very high level languages	Refine, Use.It
- Specification by form filling and screen painting	Netron/CAP, Transform
- Language sensitive editing	R1000, Genera
- Object management	R1000, Refine
- Reverse engineering of existing applications	Bachman/series
- Knowledge-based techniques	Bachman/Series, Refine, Spectrum

Some of this class of tool also address the problems of team-working by providing multi-user capability, but there seems to be little evidence of a general move towards Ipse type support. There may be sound marketing reasons behind this decision of course. Some instances of the tools in this class are shown in Table 6.

Integrated project support environments

This is a relatively small cluster at the moment which appears to have had its origins in the DoDs Stoneman report in 1980 specifying the Apse. (Ada programming support environment). An Apse consisted of a database, an interface to both users and tools and some tools. This database was a project filesset intended to store information about the history and purpose of objects, rather than their content. Object and configuration management was therefore an important issue in an Apse.

Product	Supplier
bachman/series	Bachman Information System Inc
Knowledge Build	Cullinet
Use. it	Higher order Softwear Inc
Netron/CAP	Netron Inc
R1000	Rational
Refine	Reasoning Systems Inc
Spectrum	Software A&E
Genera	Symbolics
Transform	Transform Logic Corp
Vax Cobol Generator	DEC
Pacbase	CGI Systems Inc

Table 6. Instances of Code Generator Case products

In Europe, we took out the Ada exclusivity and it became the Ipse, first defined by Alvey. Also, there was a switch from Programming support to project support which emphasised the life-cycle project management and control activities.

In effect, an Ipse provides a framework (operating system, if you like) which can be populated with tools connected to public tools interfaces. This is still an emerging technology and there is a sub-market developing for tools for integrating tools within an Ipse. The PCTE programme of ESPRIT (portable common tools environment) possibly has the potential for this aspect of Case tooling since it defines a de facto standard interface between a framework Ipse and Case tools. Many manufacturers are already building both Ipse and tools which conform to this interface standard. In ICL an early form of Ipse was developed in the early '70s called CADES (Computer Aided Development and Evaluation system – see ICL Technical Journal Vol. 2 Issue 1 for May 1980 for a good description of this).

Of major significance is the facility for Ipse to provide a means by which a chosen methodology and standards can be imposed across the entire project structure, rather than just at the technical or 'point' component. For instance, life-cycle models are frequently supported by phase control procedures which act as break points between the phases. The product must satisfy some phase acceptance criteria before being allowed to progress to the next phase. In an Ipse controlled development world this process can be significantly enhanced by (say) ensuring that appropriate quality assurance programmes have been carried through. Some instances of these Case tools are shown in Table 7.

The benefits from Case

There is a temptation at this point to extend the above descriptions of the attributes of Case to include a discussion of the technology and architecture of these classes, but maybe this is best left to another article.

Products	Suppliers
Software Backplane	Atherton Technology Inc
BIS-Ipse	BIS Applied Systems
Genos	GEC Software
Istar	Imperial Software Technology
Maestro	Philips
Perspective Kernal	Systems designers plc
EPOS	Systematica Ltd
PCTE	Bull, ICL, GEC, Olivetti, Nixdorf, Siemens, (ESPRIT programme)
Iipse 2.5	STC Tech. ICL, Dowty Electronics, Rutherford Appleton Labs, Plessey Research, British Gas Manchester Univ. (Alvey)
Eclipse	Software Sciences, CAP, LBMS Ltd. Universities of Lancaster, Strathclyde, and University College of Wales (Alvey)
Aspect	Systems Designers, ICL MARI Advanced Electronics, Universities of York and Newcastle. (Alvey)

Table 7. Iipse based Case tool examples

Instead I want to discuss the benefits that appear to be derived from the present generation of these tools. Presently, the market is still in its infancy, both in the range and scope of products available. Much of the initiative is coming from spin-off companies and the list seems to grow daily. So far, the use of Case is very spotty and very few large companies have committed themselves to this technology, but they will. The installed base is presently estimated at between 20 000 and 30 000 units and its a real growth market. That 20/30K units in 1988 in the US and UK is forecast to rise to something like 750K installed during 1995. (Interestingly, about 200K of that figure represents workstations attached to Ipses.) This would bring Case, in value terms to around 8% of the total software market at that time, but think of it as representing around 25% of the present software investment. Because of the relatively small base of installed systems not a lot of figures are available. However, lets see what we've got.

At an address to the fourth Index Technology User conference, Paul Hessinger pointed to the growing gap between software and the ability of software builders to deliver. He estimated this gap to be 4% compounded annually. (*This figure differs from the one I quoted at the front of this paper, so let's say between 4 and 8% and not argue too much!*) He clearly believes that if ever there was a requirement looking for a solution, this is it. Richard Carpenter, also of Index Technology, said that a survey of the Case user market showed the following impact:

- 10% Very favourable
- 80% preliminary results favourable
- 10% no impact
- (None said negative impact)

He also pointed out that there are four levels of proficiency in their use:

beginners	drawing of graphic diagrams and documentation
intermediate	uses dictionary, produces reports
advanced	user analysis, verify consistency, completeness, compliance, rulesets
mastery	re-uses previous design and dictionary objects

Really, the benefits from Case can be separated into:

- benefits from software engineering methods themselves
- benefits from using Case to automate those methods

It is rarely possible to properly quantify the overall benefit due to the lack of adequate metrics. The key point is to acknowledge that software engineering methods themselves can result in significant increases in productivity. British Aerospace provides an instance of this where the use of methods substantially improved productivity. In theory, it is possible to implement these methods purely manually, but invariably, on any reasonable scale project, the amount of paperwork generated can swamp the management of the project. Case tools absorb a great deal of this overhead and enhance the potential for introduction of these methods. The two approaches are therefore interdependent. Here are some of the reported benefits from using Case.

AT&T "Improved productivity accuracy and morale". Reduction of data elements from 12 000 to 900.

BDM Corp Direct productivity gain of two during coding and testing phases (Using Symbolics workstations).

British Aerospace Fivefold productivity increase in avionics and aircraft utility systems software. Major changes reduced in number by a factor of six. Achieved with tools from various sources used in BAe's Safra methodology.

Carrier Corporation For a new application development: 45% overall productivity improvement, 79% decrease in development time, no maintenance for nine months (using DesignAid, Joint Application Design and Telon).

DuPont Six to one productivity gain (Using Application Factory code generator instead of a 3GL).

Marine Midland Bank Development of major banking system (for securities clearance) in two years instead of three and a half years (Using Excelerator and Telon).

Norma Industries IS expenditure is 0.6% of turnover versus around 5% for the manufacturing average (Using Netron/CAP).

Philips Artillery command software completed on schedule, using Rational by engineers with no previous Ada experience. Rational system estimated to have paid for itself twice over within seven months.

Symbolics Have written four compilers (including Ada) in 24 person years - well above the industry average.

TRW 100% productivity gain on two projects, 200% on a third, using a "first generation Ipse" (Unix plus tools).

In a number of cases there has been a saving in code of between 5 and 10 fold reported. As a final word (or picture) on this topic let me offer you Barry Boehm's productivity improvement tree, from TRW, in Fig. 3.

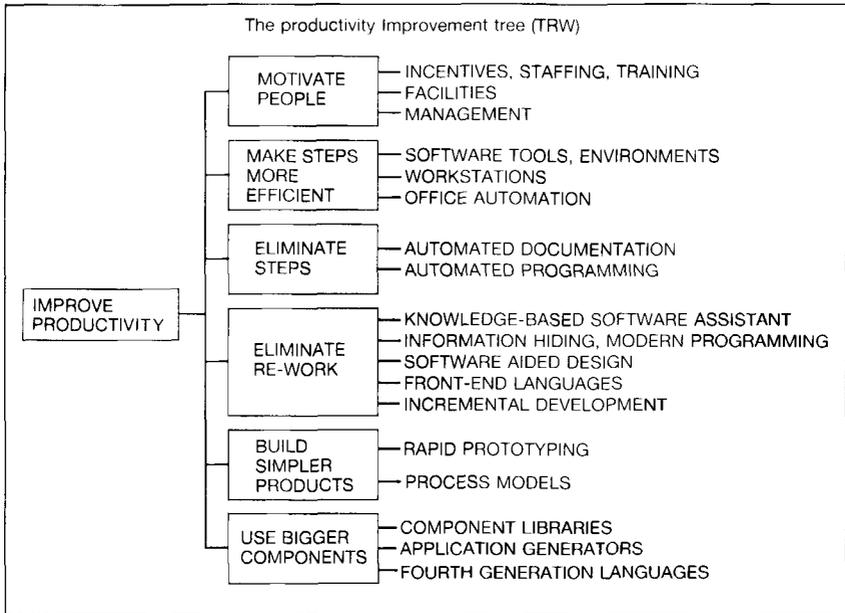


Fig. 3

Summary

No, I'm not about to embark on a long winded summing up. My personal conviction is that Case will become as essential to the IS development team as CAD now is to designers of advanced microelectronic components. There will probably be a series of shake-outs of the many builders of Case tools in the world today as the user population increases and matures in its appreciation of these tools. I suspect one of the key changes these tools will bring about however is more to do with people than machines or software. It is possible that there will be a gradual shift in skills requirements as the potential for automating large chunks of the software process become more widely realised through the evolution and spread of Case technology. We can expect to see that employers who are using Case will have an easier job in attracting top quality staff than those who still offer today's technology. In the end, it is the combination of good people and good tools that will have the greatest effect on raising productivity.

Bibliography

- Computer-aided software engineering. An Ovum report, by Julian Hewett and Tony Durham. Pub. Ovum Ltd. London 1987.
- The STARTS Guide, second edition 1987. Prepared by industry with the support of the DTI and NCC.
- Software engineering economics, Barry Boehm, Prentice-Hall 1981.
- The costs and values of Case, Capers, Jones, Software Productivity Research Inc. 1987.
- Software engineering strategy, Alvey Directorate. 1983.

The UK Inland Revenue operational systems

E. Wilson

Technology Group Manager, Inland Revenue Development Centre, Telford, Shropshire

Abstract

The computer system developed to support the operational needs of the UK Inland Revenue Service is one of the largest in the world, and the needs among the most demanding. This distributed system, based on a number of ICL Series 39 Level 80 mainframes, is now dealing with about 30 million tax cases. The following note is intended as a forerunner to a future fuller paper; it gives an outline of the main components of the complete system, focusing on the validation and integration procedures used, the system management practices and the underlying organisational issues.

Background: the major system

There are currently four major Inland Revenue computer systems supporting Operational needs. These are:

- 1 COP (Computerisation of PAYE). This system supports the administration of Pay-As-You-Earn for about 25 million UK employees. Implemented nationally between 1982 and 1988, it is an online system supporting 26 000 terminals, installed in some 700 district taxation offices, generating peak online traffic of over 200 messages per second. It is implemented on 12 ICL Series 39 Level 80 mainframes installed in 11 geographically distributed Processing Centres (PCs).
- 2 CODA (Computerisation of Schedule D Assessing) administers Schedule D taxation for some 4.5 million cases. It was implemented nationally between 1986 and 1988 as an extension of the COP system, requiring 10 000 additional terminals.
- 3 Accounts Office Systems are batch based systems supporting tax collection and accounting, to be progressively replaced from 1990 by a major new system called BROCS. The systems are implemented on 2 Series 39 Level 80 mainframes in 2 Accounts Offices (AOs).
- 4 NTS (National Tracing Service) offers facilities to trace employees or employers by name and address fragments. It is implemented on a single Series 39 Level 80 mainframe.

These systems interwork over a private X.25 network; the overall network architecture is shown in Fig. 1.

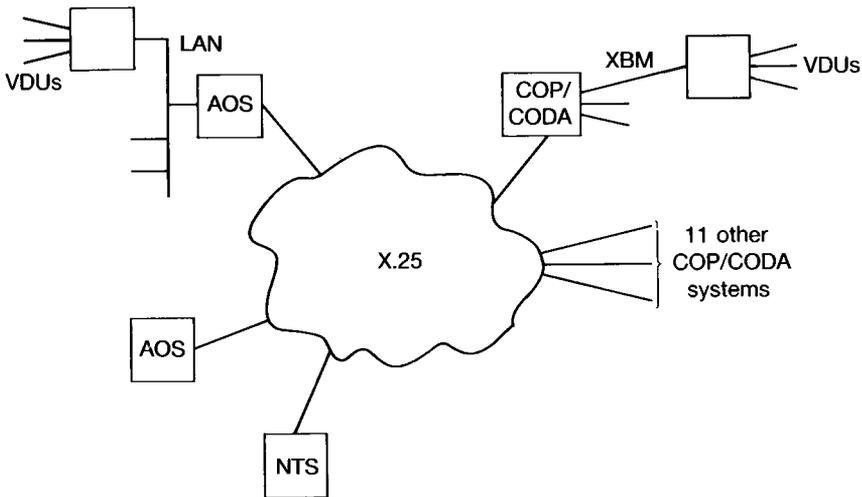


Fig. 1 Inland Revenue Network – Operational Systems

System validation

All IR Operational systems are subject to significant levels of planned change, largely to accommodate changes in Government legislation or Departmental working practices. At the same time the hardware, system software, and management tools are themselves being improved. Change within the application systems is controlled by rigorous configuration management procedures which themselves support a hierarchy of functional tests from the unit level up to a User Acceptance test. Analogous practices control change within the environment in which the application will operate.

Two principles underlie the approach to change in this environment. The first is to *minimise variety*. All operational systems are constrained to have a common software baseline; changes needed for one system (or one instance of one system) are, after due validation, replicated everywhere. A similar approach underlies the hardware baseline. The number of different types of hardware components is minimised; standard configurations are constructed for different bands of capacity; common hardware modification levels and microcode levels are imposed.

The second principle is *evolution not revolution*. Changes to software are introduced across a stable hardware base and vice versa; supplier support of this policy is demanded. Both software and hardware changes are introduced in a limited number of individually validatable packages updating their relevant baselines.

These principles dictate the approach to changes to the systems environment. Validation of such changes is effected through specific product validations, clone tests, and load tests. Specific validations confirm that new system software is non-regressive and conforms to specification. Successful collaborations with suppliers to accelerate product validation have been entered into with mutual benefit. Clone testing exercises a 'clone' of one of the live system instances within its exact hardware environment prior to release of a package of system or application changes. Load testing exercises a realistic abstraction of a live system instance under conditions of full online load, generated by a network simulator working across a real network.

System management

Systems management of the Operational systems is principally concerned with

- Problem management. The progression of perceived problems from whatever source through to timely resolution.
- Fault management - alerting, diagnosis, investigation, and recovery.
- Network configuration management.
- End system software management - software distribution, invocation, regression.

From the outset the principle of centralised management and support has been adopted, both because of its appropriateness to replicated identical system instances and in order to focus scarce skills. The central support unit at Telford Development Centre uses an extensive collection of support tools, provided by suppliers or developed in house, to effect the management of the end systems. The timely availability of ISO management standards and conformant products is seen as a major requirement, in order to meet future demands of increased terminal functionality, system interworking, and complexity. In the absence of such standards, the Revenue has collaborated with suppliers to provide generic, marketable, solutions.

Support interfaces

Centralised problem management is complemented by the provision, by ICL and other major suppliers, of a single point of contact for support regardless of fault type or location. ICL, as the supplier of both mainframe and terminal hardware has its service management centre co-located with the Revenue's problem management group. Service levels provided by suppliers are contractually determined and closely monitored.

Achievement

One important measure of the effectiveness of the various procedures outlined is the availability of the computer services to the end user. A terminal availability of 98.67% was achieved during 1987, covering all

reasons for unavailability including mains power failure and operator error. A target of 99.2% was adopted for 1988.

Conclusion

Inland Revenue developments in the field of IT are dedicated to producing solutions to business needs. Implicit in these solutions are high levels of system reliability. Successful tools and procedures have been adopted to achieve this across a network of complex replicated systems. We look to our suppliers to continue to improve product reliability and management tools, and thus to allow us to focus on the IT solutions themselves.

La Solution ICL chez Carrefour a Orleans

Y. Pisigot

ICL France, 78140 Velizy-Villacoublay

Abstract

CARREFOUR, is the world wide largest hypermarket chain, with 100 hypermarkets: 71 in France, 25 in Spain, the remaining 4 in South America. The latest of Carrefour, France opened in April 1988 and is entirely managed by a system running on ICL CLAN and DRS equipment. The sales area is 6000 square metres (60 000 square feet), the stock held is about 100 000 items, there are 34 check-out tills with optical scanning and EFT equipment, and about 7000 customers use the store daily, spending an average of 120 F (about £11) each; the number of items sold daily is about 200 000. The system has direct links to the banking system through the French national digital data transmission network TRANSPAC; it also manages the Carrefour private card (carte PASS). The paper gives a summary of the architecture, function and performance of the store management system.

1. Introduction

1.1. Le système d'encaissement et de gestion installé à CARREFOUR ORLEANS a pour objectif la gestion complète du magasin. Il traite les flots d'informations circulant à l'intérieur du magasin et permet l'échange de données entre le magasin et différents partenaires extérieurs:

- siège
- tournisseurs
- réseau bancaire
- etc ...

1.2. Schéma de principe

1.3. Plusieurs catégories de personnels sont en contact quotidien avec le système

- les caissières
 - les responsables encaissement
 - les responsables du coffre
- } Pour les fonctions "encaissement"
G.M.S.*

* General Merchandising System.

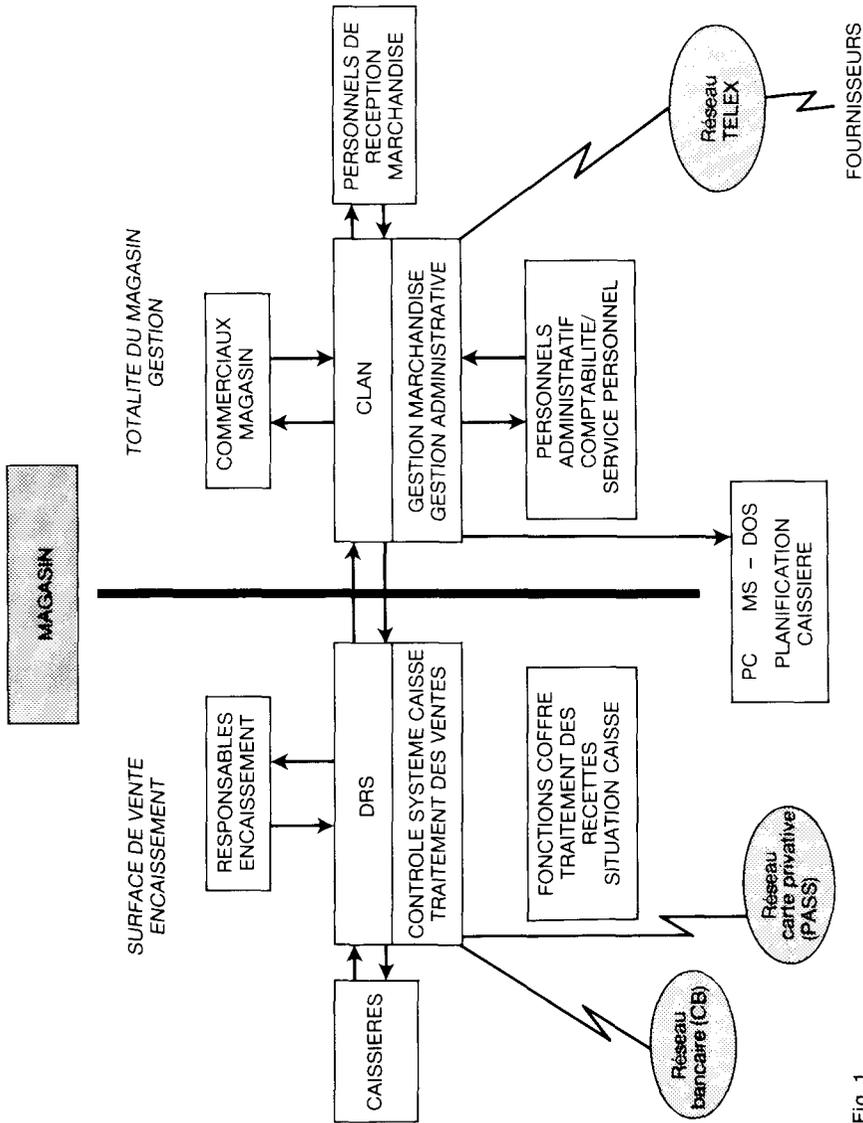


Fig. 1

- les commerçaux
 - chefs de rayon
 - secrétaires d'achat
 - les personnels de réception marchandise
 - les personnels administratifs (comptabilité)
- } Pour les fonctions "gestion" sur le CLAN 7

2. Configuration generale

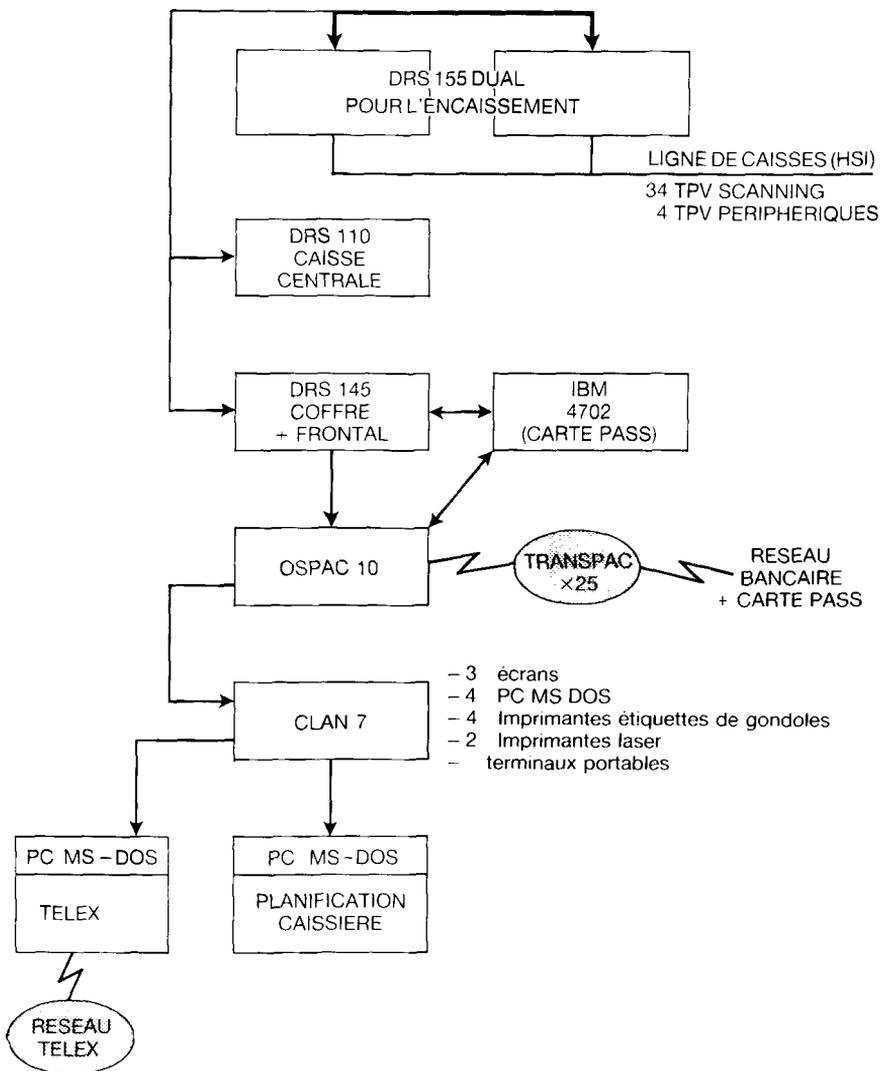


Fig. 2

3. Encaissement. Systeme G.M.S.

3.1. Fonctions POS

- Passage des articles (scanner)
- saisie des ventes
- encaissement par moyens de paiement
 - Carte Bancaire – connexion au réseau bancaire
 - Carte PASS – signature électronique sur PinPad Crouzet
 - connexion via 4702 au réseau Carte PASS
 - gestion comptant/crédit
 - Chèque – lecture/encodage CMC7 via Crouzet ELC200
- avantages financiers (remise) aux clients privilégiés
- retrait d'argent (fonction ALEX – Carte PASS)

3.2. Fonctions caissière

- gestion du caissier flottant
- tiroir personnalisé
- prélèvements

3.3. Fonctions coffre

- saisie recettes/dépenses
- suivi des caissières
- mouvements d'argent

3.4. Fonctions fichiers

3.4.1. Tailles

utilisateurs	300
emplacements	50
moyens de paiement	10
articles	100 000
remises/offres/réductions	99

3.4.2. Articles

Environ 50 000 articles (sur les 100 000 prévus) sont actuellement introduits sur le système G.M.S.

La répartition FAM/disque est faite automatiquement à partir du fichier Département (environ 30).

Les articles à fort passage – tous produits de grande consommation – sont en FAM (40 000).

Ceci permet un passage caisse sans ralentissement pour la totalité des articles.

3.4.3. Remises/offres/réductions

Tous les avantages financiers sur ticket sont appliqués – en pourcentage article ou département – de façon automatique.

Deux possibilités:

- application à l'ensemble des clients
- application uniquement aux clients privilégiés (carte Pass)

3.5. Fonctions transmissions

3.5.1. Accès direct via X25 au réseau bancaire

- demandes d'autorisation au GIE Carte Bancaire
- traitement des listes d'opposition (listes noires)
- télécollecte des remises bancaires (E.F.T.)

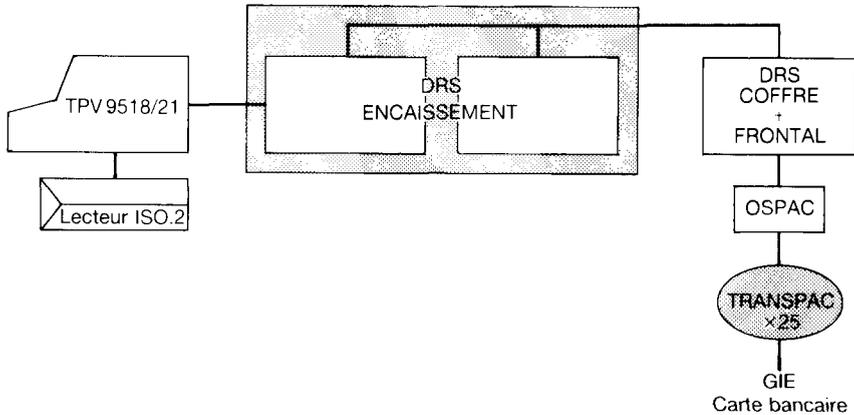


Fig. 3

3.5.2. Accès direct via X25 au serveur Carte PASS

- contrôle du code confidentiel (sur 4702)
- traitement crédit/comptant
- fonction de retrait d'argent (ALEX)
- fichier clients magasin (dans 4702) ou extérieur (centre serveur)

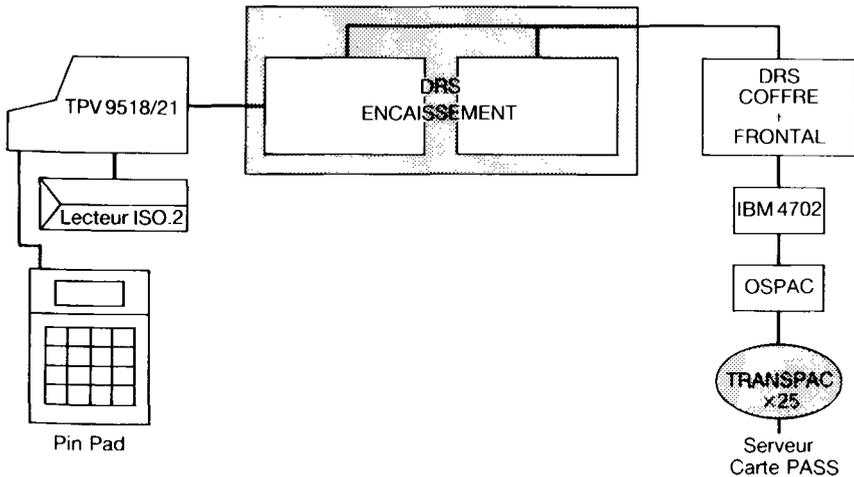


Fig. 4

3.5.3. Accès interactif au système Back Office via X25/ADI

- pour mise à jour article sur G.M.S.
- transfert des éléments de vente en fin de journée

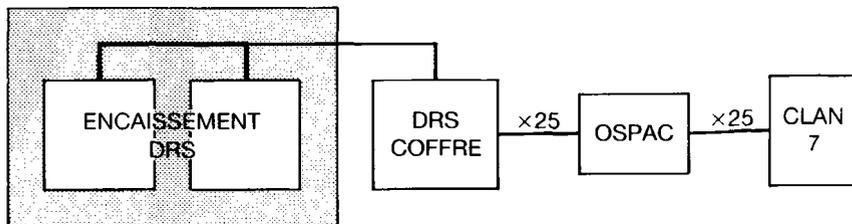


Fig. 5

3.6. Configuration G.M.S.

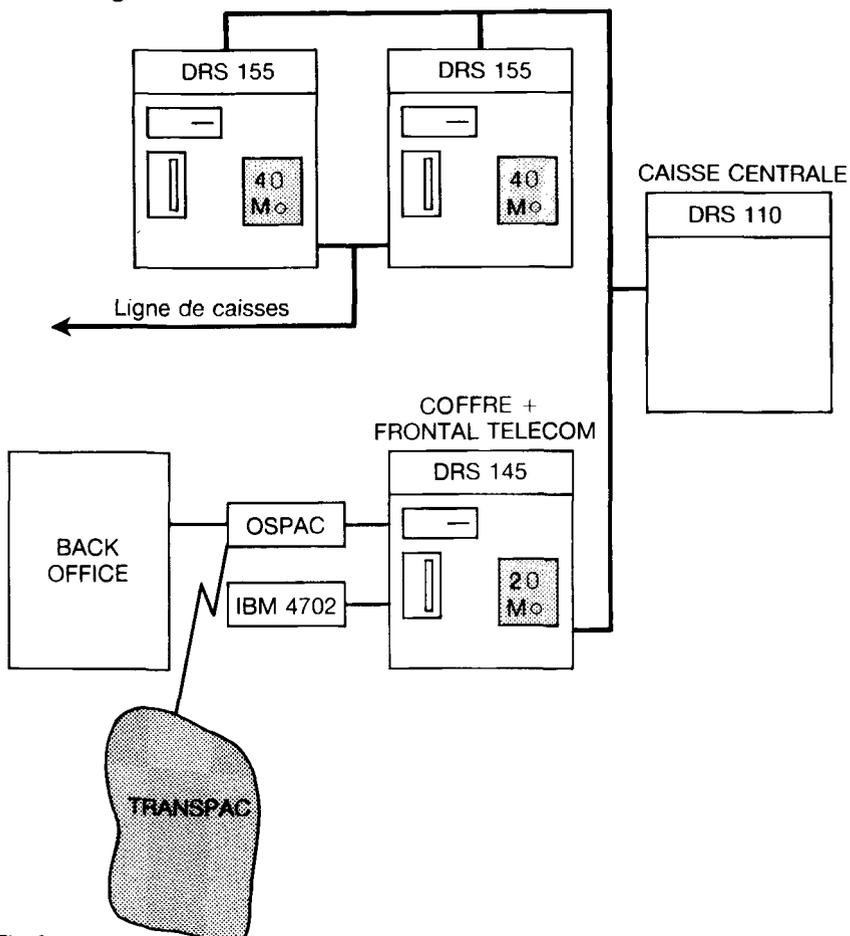


Fig. 6

4. Gestion magasin

4.1. Synoptique des fonctions

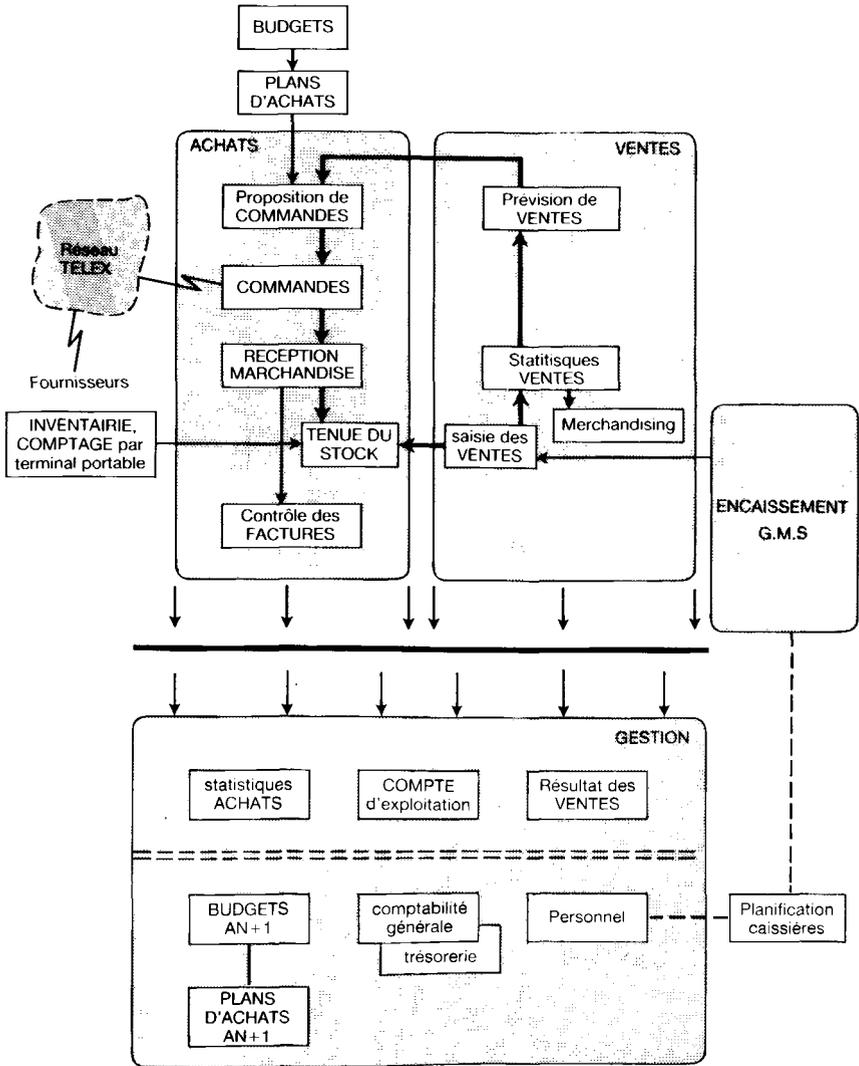


Fig. 7

4.2. Fonctions de gestion magasin

4.2.1. Gestion marchandise

Cycle achat

- proposition de commande
- passation de commande: en direct par télex ou par courrier
- réception marchandise
- traitement des factures
- tenue des stocks – avec inventaire et comptage
- maintenance des fichiers de base
- promotions
- ristournes de fin d'année
- statistiques achats

Cycle vente

- saisie des ventes (système GMS)
- gestion unitaire – merchandising
- étude assortiment
- analyse concurrence
- prévisions de ventes
- statistiques de ventes
- tableau de gestion
- analyse chiffre d'affaires – historique – objectifs

4.2.2. Gestion administrative comptabilité générale/trésorerie compte d'exploitation etc ...

4.3. Fonctions transfert

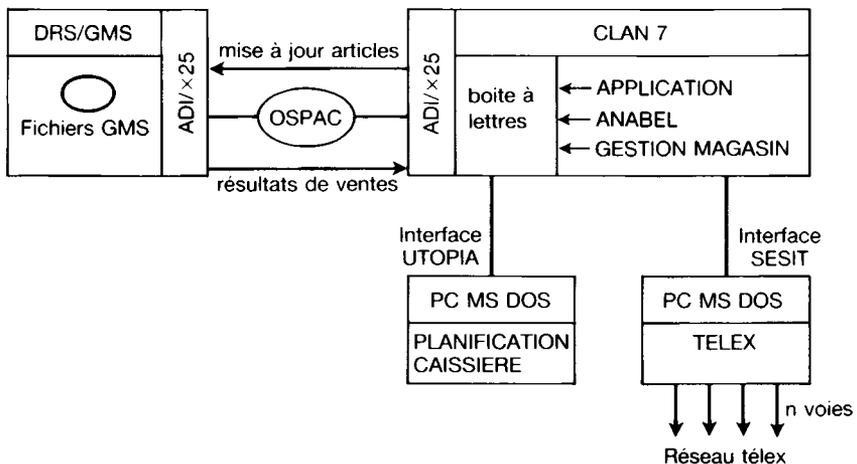


Fig. 8

4.3.1. Schéma général

Le CLAN est maître dans cet échange et initie tous les transferts avec le système GMS

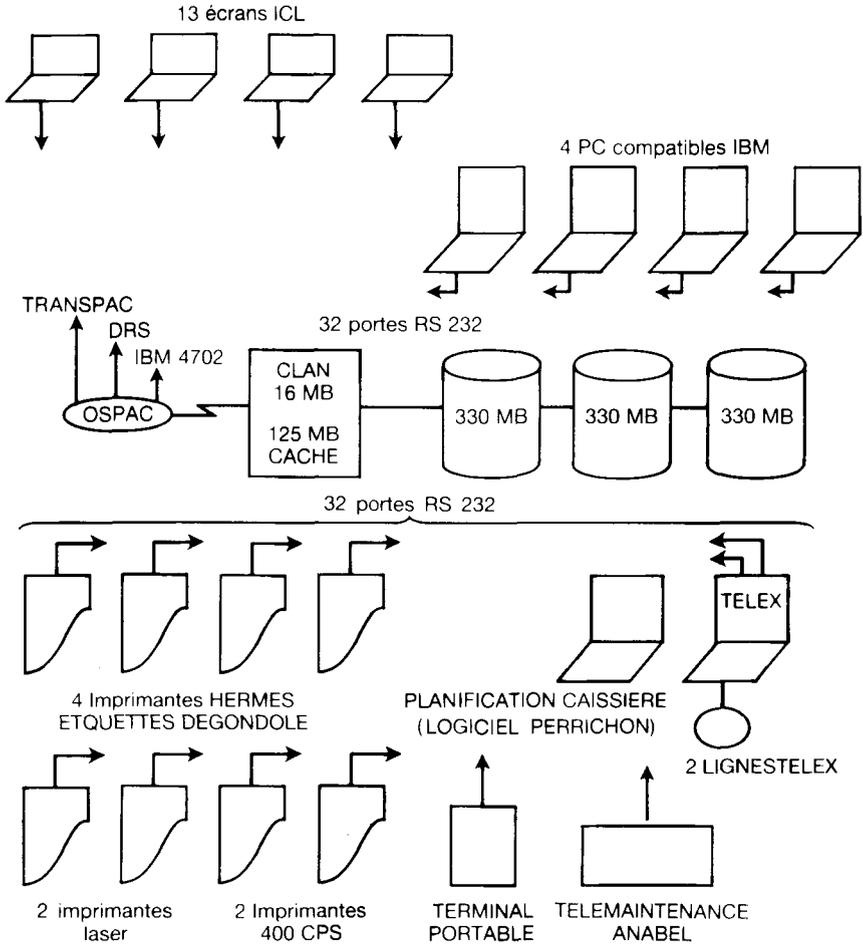


Fig. 9

4.3.2. Mise à jour articles

Toutes les opérations de maintenance du fichier articles sont réalisées sur le CLAN, avec transfert vers le système GMS:

soit dans la journée – pour application immédiate

soit le soir – pour application pendant le traitement de nuit GMS

4.3.3. Résultats de ventes

Le transfert des résultats de ventes est essentiel pour le système de gestion (saisie des ventes unitaires)

L'ensemble du traitement sur le CLAN: proposition de commande, différents tableaux de bord, etc. n'est possible qu'après ce transfert.

Un mécanisme est mis en place pour que l'information soit transférée dès que disponible (entre autre, avant tout traitement GMS de type JOIN).

En cas de rupture de la liaison, toutes les informations sont archivées sur DRS jour par jour et le transfert est automatique.

4.3.4. Planification caissière

Cette fonction est exécutée sur un PC MS DOS directement à partir des informations fournies par le système GMS; le CLAN sert donc de boîte aux lettres entre GMS et le PC.

Toutes les informations sont disponibles jour par jour sur le CLAN et le PC va les chercher quand elles lui sont nécessaires.

4.3.5. Réseau télex

Cette liaison est utilisée pour transférer les commandes après validation par un responsable commercial, aux différents fournisseurs. Cette fonction est réalisée par un PC connecté au CLAN et au réseau télex (4 voies simultanées) via un interface agréé.

4.4. Configuration CLAN

5. Partenaires dans la solution Carrefour

ICL	- encaissement DRS 155/DRS 145 GMS 9518:21
CROUZET	- matériel gestion CLAN 7 écran - lecteur encodeur CMC7 pour chèque - PIN PAD pour code confidentiel
ANABEL	- logiciel gestion magasin (CLAN)
PERRICHON	- logiciel de planification caissière sur PC connecté
SESIT-ITAB	- connexion au réseau télex
AXIS DIGITAL	- connexion PC MS DOS à UNIX CLAN 7 (UTOPIA)

6. Caractéristiques du Magasin Carrefour Orleans

- ouverture le 27/04/88
- directeur: M. DECLEMY
- surface de vente: 6000 m2 (magasin de centre ville)
- 250 personnes employées

- 34 TPV en ligne de caisse
- 4 caisses périphériques
- 7000 clients/jour
- mouvement de 12/1400 articles/jour
- panier moyen 120 F
- objectif 250 000 articles sur le CLAN (80 000 aujourd'hui)
100 000 articles sur GMS (50 000 aujourd'hui)

A Formally Specified In-Store System for the Retail Sector

Val Jones

Department of Computing Science, University of Stirling,
Stirling FK9 4LA, Scotland

Abstract

The paper reports work completed under the auspices of Alvey Software Engineering Project SE029, *Use of Functional Programming as a design and prototyping methodology for Intelligent Business Systems*. This project involved collaboration between, the Department of Computing Science, University of Stirling, STC Technology Ltd. and the Institute of Retail Studies, University of Stirling.

Retailers are well-established users of computer technology as aids to traditional managerial functions such as financial management, inventory control, space allocation and physical distribution. New technology also opens up the possibility of extending new services to customers. Here we present a design for an integrated supermarket system in which the functions of space allocation and inventory management are handled automatically. Customer services are also provided – a home browsing and teleshopping service and an in-store information service. The small databases used as examples are taken from food retailing, but the system is equally applicable to other retail areas such as electrical goods, DIY, garden centres and so on.

The complete system was formally specified; a few extracts from the specification are given in Appendix 1, to illustrate style and structure. The specification language used (**me too**) is executable and a prototype of the retail system runs on a microcomputer.

The purpose of this exercise is to demonstrate that formal methods can be applied to problems drawn from the real world. Here certain simplifying assumptions have been made concerning the problem domain, but the author believes that the result demonstrates the feasibility of the approach.

1 Introduction

The large retail chains are well-established users of conventional computer technology to handle traditional managerial functions such as financial

management and inventory. Many small retail outlets now use off-the-shelf software running on microcomputers to perform similar functions. The adoption of barcoding standards by producers means that wider use of EPoS and EFTPoS systems (electronic point of sale, electronic funds transfer at point of sale) is in prospect. In addition, many large retailers are increasingly investing in new technology to aid in planning tasks such as space allocation and physical distribution. New technology has also opened up the possibility of extending new services to customers.

Presented here is a design for an integrated in-store supermarket system which provides four facilities – two traditional management functions and two novel services to customers. The two traditional applications are space allocation and inventory management. The customer services provided are an in-store information service and a computer-based home browsing and shopping service. These represent four of the application areas identified in [6].

For the store manager

The **space allocation** application automatically allocates products to shelves in a supermarket. The supermarket may have any arbitrary layout. The space allocation system generates a plan for distributing products across shelves so that related products are shelved together. This plan can be inspected and modified by the manager. The **automatic perpetual inventory** application exploits EPoS and barcoding to continually monitor shelf stock and store room stock. Warnings are given of impending shelf stock-outs and, when stocks fall to a certain level, goods are automatically reordered from the (currently cheapest) suppliers.

Customer services

The **in-store information** service is intended for large stores, where customers may require help in finding a particular product. From any location in the store the customer can request directions to a particular product. The system calculates the shortest route from the customer's current location to the nearest instance of that product, and generates directions for the customer. The **teleshopping** application allows customers to access (selected parts of) the store's database from a terminal in their home. The customer can browse through the product information, decide what they require, and order goods. The transaction will be effected providing the customer's credit is good. If so, the customer's account is debited by EFT, and appropriate updates are made to inventory, sales and orders to suppliers.

The examples which follow are taken from food retailing. The system, however, is equally applicable to other retail areas such as hardware, electrical goods, DIY, garden centres, and so on.

The retail system was designed using the **me too** method [1–5]. This method

uses the techniques of formal specification and rapid prototyping in the design stage of the software development process. Here the *functionality* of the system (what it does), rather than the user interface, is being prototyped. The primary output from the **me too** method is the formal specification, whose purpose is to act as a design document – as a guide to the implementation and testing of the eventual software product. The system is specified in the **me too** language; a sample of the specification is presented in Appendix 1. The **me too** language is executable and a prototype system runs on a microcomputer.

The purpose of this exercise is to demonstrate that formal specification can be used to capture and express, economically and unambiguously, the important aspects of a “real-world” problem. Proponents of formal methods claim that the incorporation of these techniques into the software engineering process increases the reliability, correctness and maintainability of software products, and can decrease the costs and time involved in production and maintenance of software.

Certain simplifying assumptions have been made concerning the problem domain. The specification does not attempt to cover all operational aspects of a retail outlet; nor does it deal with the full complexity of the functions which it does handle. Moreover, the store is treated in isolation, whereas some aspects of the design would be more appropriate to large multiples than to the small retailer’s operation, where only one outlet is involved. The author believes, however, that there would be no obstacle to extending and generalising the specification in order to address these complexities, and hopes that sufficient complexity has been retained here to demonstrate that a formal approach could be both feasible and fruitful in a real industrial project.

2 The Store

The store consists of a collection of databases storing information on inventory, suppliers, orders, sales, products, barcodes, customers and store layout. Each component of the store is treated as a separate data type, and a set of operations on that type is defined. For example, there is a type Inventory, and a set of operations, including constructors and selectors, are defined for objects of type Inventory. A data object of type Inventory is built, and combined with the other objects (of types Suppliers, Orders, etc.) to build an object of type Store. For each component of the store, there is a corresponding module in the specification.

Each of the four applications makes us of the major type, Store, and to a subset of the other types. The types are listed below:

- St Store
- I Inventory database
- S Supplier database

- O Orders (to suppliers)
- SI Sales (i.e. accumulated total sales in pounds)
- P Product information
- B Barcode table (associations between barcodes and product descriptors)
- C Customer database (customer identifiers and credit ratings)
- L Layout of store

The store components in turn make use of various libraries of extensions to the predefined operations on basic **me too** types, such as relations and sequences. These components are as follows:

- N extensions to operations on numbers
- T extensions to operations on sets
- R extensions to operations on relations
- G operations on bags (bags are sets with repetitions)
- Q extensions to operations on sequences

Figure 1 shows which components are used by each application:

Application	Application-specific components										Generic components					(b)
	St	I	S	O	SI	P	B	C	L	N	T	R	G	Q		
(a) Space Allocation	X	X				X	X		X				X			
(b) Perpetual Inventory	X	X	X	X	X	X	X						X			
(c) Instore Information	X								X		X	X				
(d) Teleshopping	X	X	X	X	X	X	X	X					X		X	

Fig. 1 Components used by the retail applications

The last column shows that some of the operations in the Automatic Perpetual Inventory application are reused in the Teleshopping application.

Each application consists of the set of specifications defining the components used by that application, together with the application specification itself. Each application specification, therefore, should be read in conjunction with the specifications of the components which it imports. For example, the Space Allocation component uses the Store, Inventory, Products, Barcode Table and Layout components. An understanding of this application therefore depends on reading the specifications of these components.

The type Store represented by a record containing eight fields.

Store = record (inventory : Inventory,
suppliers : Suppliers,

orders : Orders,
sales : Sales,
products : Products,
bartable : Bartable,
customers : Customers,
layout : Layout)

The record declaration is shown below.

Store : : inventory, suppliers, orders, sales, products, bartable, customers, layout;

When the declaration of the record type **Store** is input to the **me too** system, record constructor, selector and update operations are automatically created for objects of type **Store**.

Each field type refers to a component of the system which is specified in a separate module. Instances of that type can be constructed using the operations provided in that module.

3 The Store Components

There are many decisions which could be made concerning where information is to be distributed across different parts of the system. The choices made below are not the only possible ones.

The specification of the retail system focusses on the functionality of the system, and not on the user interface. The interactions with the **me too** shell shown below are meant to illustrate this functionality. An implementation, as opposed to a prototype, would incorporate appropriate interfaces such as menus, mice, touch sensitive screens, and so on.

3.1 Inventory

Each inventory entry shows the following information concerning a product stocked by the store:

ONSHELVES – quantity currently on the shelves
MINSHELF – shelf stock level at which shelves should be replenished
TOTSHELF – total shelf space allocated to this product (units of product)
INSTORE – quantity held in the store room
REORDLEV – stock level at which this product should be reordered

Inventory records are indexed by barcode. Initially the store stocks the following products:

<i>Product</i>	<i>Barcode</i>
Master Blend ¹ ground coffee (size 230g)	5000136997729
Nescafe ² instant coffee (size 50g)	5000243000204
Nescafe instant coffee (size 100g)	5000243924403
Cafe Hag ³ instant coffee (size 50g)	5000136998627
Irn Bru ⁴ (size 2 litres)	5000107000827
Felix ⁵ tuna cat food (size 400g)	5000108005445
Felix rabbit cat food (size 400g)	5000108005353

The inventory entry for each product is initially set up to allow space for 10 units of product on the shelves. The system starts with the shelves full. Shelf stock is to be replenished when it falls to five units. Additional stocks of 15 units of each product are held in the store room; (total stocks of each product, therefore, are initially set to 25). When total stocks of a product fall to 10 units, that product must be reordered from the suppliers. These figures can be adjusted as desired, for example when analysis of the store's operation results in the development of a more realistic local demand model. (Such a model could be extracted automatically using feedback from scanning data over a period of time.) The initial inventory is shown in Fig. 2.

	ONSHELVES	MINSHELF	TOTSHELF	INSTORE	REORDLEV
5000136997729	10	5	10	15	10
5000243000204	10	5	10	15	10
5000243924403	10	5	10	15	10
5000136998627	10	5	10	15	10
5000107000827	10	5	10	15	10
5000108005445	10	5	10	15	10
5000108005353	10	5	10	15	10

Fig. 2 The initial inventory

Operations from the Inventory module are used to construct a data object *inv*, which will later be supplied as the *inventory* field in the store record. The type *Inventory* is represented as a map from barcodes to inventory entries:

$Inventory = \text{map}(\text{Barcode}, \text{InvEntry})$

where

$\text{Barcode} = \text{Nat}$

$\text{InvEntry} = \text{map}(\text{Attr}, \text{Nat})$

$\text{Attr} = \{ \text{"ONSHELVES"}, \text{"MINSHELF"}, \text{"TOTSHELF"}, \text{"INSTORE"}, \text{"REORDLEV"} \}$

¹Master Blend is a trademark of General Foods Ltd.

²Nescafe is a registered trademark of the Nestle Co. Ltd.

³Cafe Hag is a registered trademark of General Foods Ltd.

⁴Irn Bru is a trademark of A.G. Barr plc.

⁵Felix is a registered trademark of Quaker Oats Ltd.

Four constructor operations are needed to build the inventory:

```
emptyinv : -> Inventory
mkent : Nat x Nat x Nat x Nat x Nat -> InvEntry
mkinv : Barcode x InvEntry -> Inventory
joininv : Inventory x Inventory -> Inventory
```

emptyinv constructs an inventory with no entries, mkent constructs an inventory entry, mkinv constructs an inventory with a single entry and joininv is used to merge two inventories. First an inventory entry is defined:

```
ent == mkent(10,5,10,15,10) ;
```

and ent is inspected:

```
ent ;
{ ONSHELVES -> 10, MINSHELF -> 5, TOTSHelf -> 10,
  INSTORE -> 15, REORDLEV -> 10 } ;
```

Now the inventory is constructed:

```
invt == emptyinv() ;
invt == joininv(invt,mkinv(5000136997729,ent)) ;
invt == joininv(invt,mkinv(5000243000204,ent)) ;
```

invt at this point has the following value:

```
{5000136997729 -> {ONSHELVES -> 10,MINSHELF -> 5,
  TOTSHelf -> 10,INSTORE -> 15,
  REORDLEV -> 10},
  5000243000204 -> {ONSHELVES -> 10,MINSHELF -> 5,
  TOTSHelf -> 10,INSTORE -> 15,
  REORDLEV -> 10} }
```

The other entries are added, and when the inventory as shown in Fig. 2 has been constructed, invt is in the form in which it will be combined with other objects to construct the store.

3.2 Supplier Database

Information about suppliers is held in a supplier database. For each supplier, there is a supplier code, the supplier's name and address, and his current pricelist (the *cost* price of each product). A small supplier database is shown in Fig. 3.

Supplier data will be held in an object of type Suppliers which can be represented as a map:

```
Suppliers = map(Suppliercode,pair(Name-Address,Pricelist))
```

where

```
Pricelist = map(Barcode,Price)
Price, Barcode = Nat
Suppliercode, Name-Address = String
```

Supplier Code	Name and address	Pricelist	
sc1	Jim Skullion Dock Road	5000136997729	140
		5000243000204	80
		5000243924403	155
		5000136998627	90
sc2	Arthur Daley The Lock Up	5000136997729	142
		5000243000204	60
		5000243924403	150
		5000107000827	53
sc3	Harry Sharp Abattoir Road	5000107000827	55
		5000108005445	30
		5000108005353	30
sc4	Frank Price Dodgy Blvd	5000107000827	58
		5000108005445	31
		5000108005353	31

Fig. 3 A supplier database

Price lists are constructed using the operations `emptypricelist` and `addprice`, and the Suppliers object using operations `emptysuppliers` and `addsupplier`:

`emptypricelist` : \rightarrow Pricelist

`addprice` : Barcode \times Price \times Pricelist \rightarrow Pricelist

`emptysuppliers` : \rightarrow Suppliers

`addsupplier` : Suppliercode \times Name-Address \times Pricelist
 \times Suppliers \rightarrow Suppliers

The supplier database is constructed:

```
pl1 == addprice(5000136997729,140,emptypricelist());
pl1 == addprice(5000243000204,80,pl1);
pl1 == addprice(5000243924403,155,pl1);
pl1 == addprice(5000136998627,90,pl1);
sup == addsupplier("sc1","Jim Skullion, Dock Road",pl1,
emptysuppliers());
```

The process continues until all the information shown in Fig. 3 has been included in `sup`.

3.3 Orders

As goods are sold, an orders database is built up. Figure 4 shows that an orders database consists of a number of orders to individual suppliers, and that each order shows the quantity of each product to be ordered from that supplier. Here four packets of Master Blend ground coffee are to be ordered from supplier `sc1`, and six tins of Felix cat food – five tuna and one rabbit flavoured – from supplier `sc3`.

Supplier Code	Barcode	Quantity
sc1	5000136997729	4
sc3	5000108005445 5000108005353	5 1

Fig. 4 An orders database

An orders database is represented as a map:

$\text{Orders} = \text{map}(\text{Suppliercode}, \text{Order})$

where

$\text{Order} = \text{map}(\text{Barcode}, \text{Nat})$
 $\text{Suppliercode} = \text{String}$
 $\text{Barcode} = \text{Nat}$

Initially the orders database is empty, so `ord` is defined using the operation `emptyorders`, which takes no inputs and returns an object of type `Orders`:

`ord == emptyorders() ;`

3.4 Sales

The sales figure is the cumulative total in pounds and pence representing the money taken so far for sales of goods. This is represented as a `Real Number`.

$\text{Sales} = \text{Real}$

The store starts off with a sales figure of zero: the operation `emptysales` takes no inputs and constructs a value of type `Sales`:

`emptysales : -> Sales`

This operation simply constructs the value zero; its definition is

`emptysales() == 0 ;`

Our initial sales object is defined:

`sal == emptysales() ;`

3.5 Products Database

Information on products is stored in the products database. The database here is structured to show explicitly the hierarchical relationships between products. This can be seen from Fig. 5 where different levels in the hierarchy correspond to different product categories: product groups, lines, brands, sizes and so on. The topmost node of the tree is "foods", since we are dealing exclusively with foods in this example. In reality a food retail outlet would also sell non-foods, so the database shown here would be a subtree of the full database. Three product groups are shown – beverages, soft drinks and pet

foods; only one or two brands per product group are included. In reality the tree would have a much higher branching factor. At the lowest level, each of the leaves of the tree is labelled with a product descriptor and stores information about that product. This example has seven leaves; a large superstore, however, might carry more than 20,000 products.

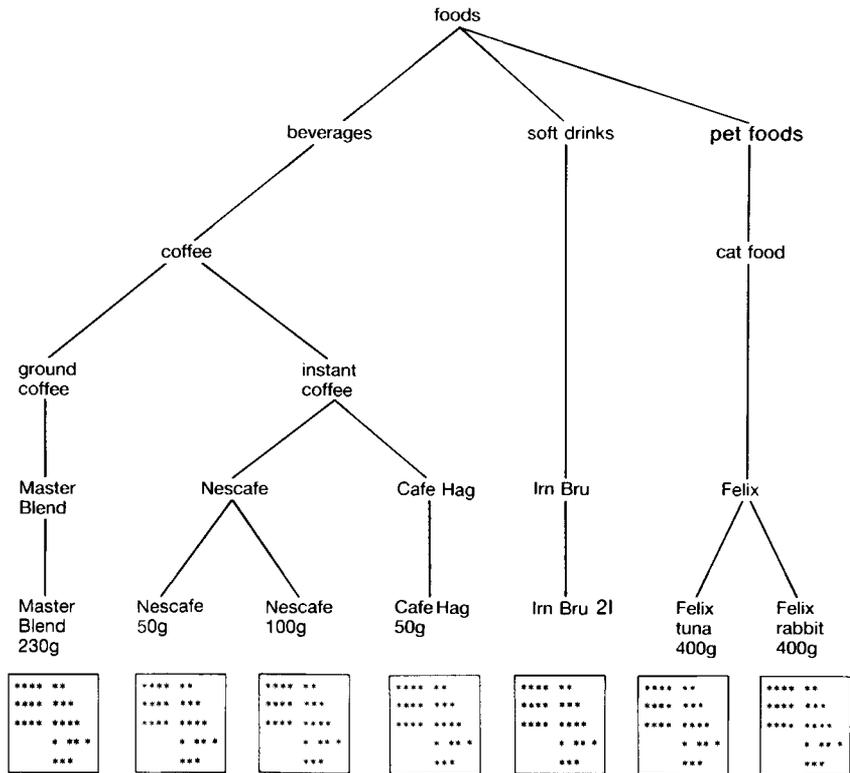


Fig. 5 The product information

This design might be implemented as a network or relational database. Alternatively, a frames system would be quite appropriate, since classes of products share certain characteristics. All the information common to a certain brand of a product could be stored higher up, for example the contents of Nescafe could be stored at the node labelled Nescafe. The lower levels (Nescafe 50g etc.) could inherit that information when required, rather than duplicating the same information at all leaves of the subtree. This would impose an overhead on retrieval operations; however the database would be significantly compacted.

A great deal of information could be stored about products; some of this information would be used by the company, and some could be accessed by customers. Here we show a sample of the information which might be stored.

Figure 6 shows an exploded view of the leaves of the hierarchy; it can be seen that the size, the selling price and the contents of products are recorded in the leaves of the product database.

The Products module recognises the following abstract objects, or types:

- Products – the product database
- Category – product classes at different levels, e.g. beverages, coffee
- Info – either the subcategories of a category, or Prodinfo
- Prodinfo – at the lowest level, the information stored about a product
- Attr – product attribute, e.g. size, contents
- Val – the value of an attribute

The following constructor operations from the Products module are needed to build the products database:

```
emptyproducts : -> Products
addproducts : Category × Info × Products -> Products
emptyprodinfo : -> Prodinfo
addprodinfo : Attr × Val × Prodinfo -> Prodinfo
```

The following representation will be used for product databases:

```
Products = map(Category,Info)
```

where

```
Category = {"foods","beverages","soft drinks", "coffee",
            "instant coffee", . . .}
Info = set(Category) union Prodinfo
Prodinfo = map(Attr,Val)
Attr = {"size","sell","contains", . . .}
Val = Int union String union set(Val)
```

Now a database is constructed, starting at the topmost node:

```
prod == addproducts("foods",{"beverages","soft drinks",
                             "pet foods"},emptyproducts());
prod == addproducts("beverages",{"coffee"},prod);
```

and so on, down to the leaves. Then the product information is constructed and added into the database for each of the leaves, as in:

```
pr1 == addprodinfo("size","230g",emptyprodinfo());
pr1 == addprodinfo("sell",169,pr1);
pr1 == addprodinfo("contains",{"coffee grounds","caffeine"}, pr1);
prod == addproducts("master blend 230g",pr1,prod);
```

3.6 Barcode table

For internal system operations, products are represented by their barcodes. Where output is for human consumption, however, (e.g. for store staff or customers), a more intelligible product descriptor is required. At times the

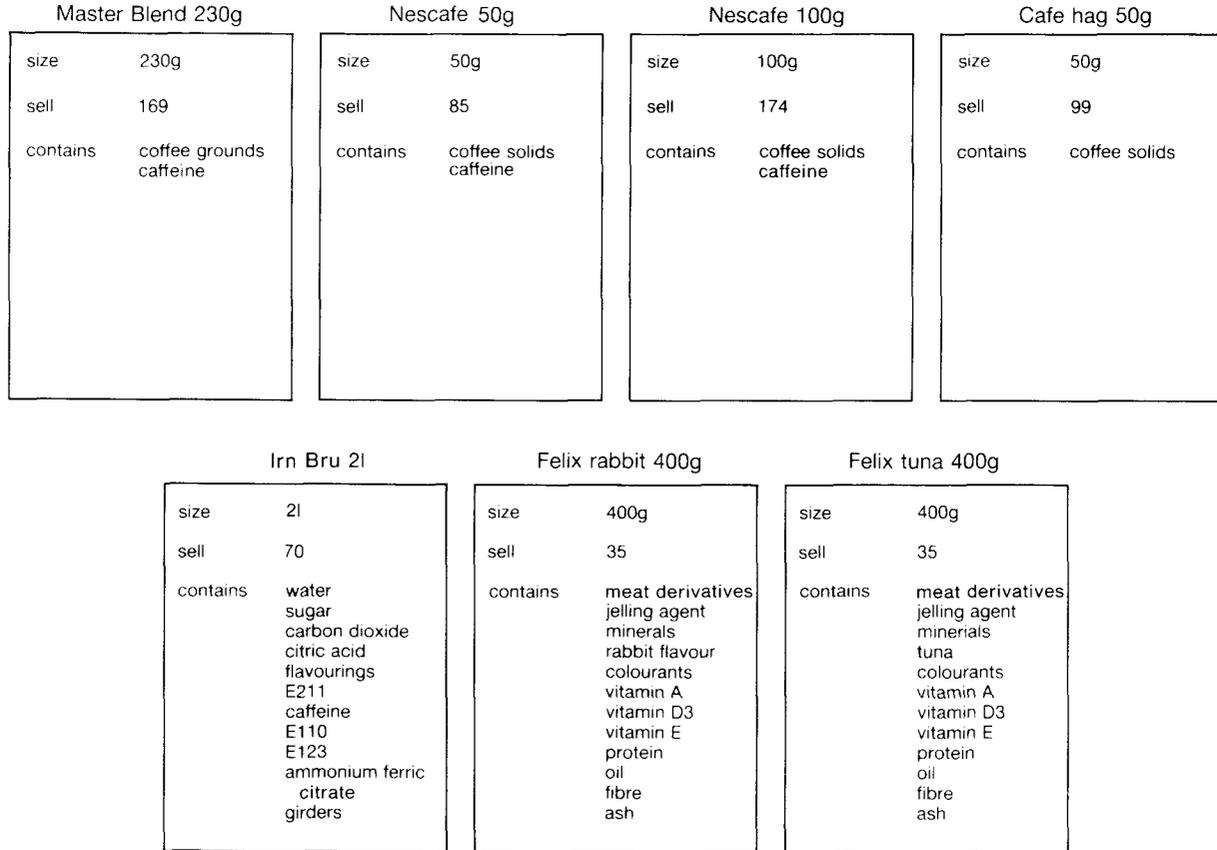


Fig. 6 The leaves of the product hierarchy

system will need to convert between product descriptors and barcodes, so we will provide a module to construct a table showing the correspondences, and to convert between the two representations. Figure 7 shows the correspondences between barcodes and product descriptors.

Barcode	Product descriptor
5000136997729	master blend 230g
5000243000204	nescafe 50g
5000243924403	Nescafe 100g
5000136998627	cafe hag 50g
5000107000827	irn bru 2l
5000108005445	felix tuna 400g
5000108005353	Felix rabbit 400g

Fig. 7 A Barcode Table

These are EAN (European Article Numbering Association) standard barcodes. The abstract objects in the Bartable module are

```
Bartable
Barcode
ProdDesc
```

and two operations are needed to build a barcode table:

```
emptybartable : -> Bartable
addbartable : Barcode x ProdDesc x Bartable -> Bartable
```

The representations are

```
Bartable = map(Barcode,ProdDesc)
```

where

```
Barcode = Nat
ProdDesc = String
```

The barcode table is built:

```
bar == addbartable(5000136997729,"master blend 230g",
                  emptybartable());
bar == addbartable(5000243000204,"nescafe 50g", bar);
```

and so on.

3.7 Customer Database

For EFTPoS transactions, a database showing customers and their credit ratings is needed. Our store has two customers registered, Mac and Morag, with ratings of 1000 and 2000 pence respectively. Figure 8 shows this customer database.

In the Customer module, a customer database is specified as a map:

Customer identifier	Credit
mac	1000
morag	2000

Fig. 8 A customer database

Customers = map(CustId,Credit)

where

CustId = String

Credit = Int

The database is built using two operations:

emptycustomers : -> Customers

addcustomer : CustId x Credit x Customers -> Customers

In our case, the customer database **cust** is constructed as follows:

cust == addcustomer("mac",1000,emptycustomers()) ;

cust == addcustomer("morag",2000,cust) ;

3.8 Store layout

The store manager enters the details of the layout of the store and its shelving to the system. This information will be used by the space allocation application, and also by the in-store customer information service. Suppose the overall dimensions of the store are 23×5 , and there are four shelf fittings. The space occupied by the shelves can be given by the coordinates of the end points of the shelf:

- shelf 1 (1,1) to (23,1)
- shelf 2 (2,3) to (21,3)
- shelf 3 (2,5) to (21,5)
- shelf 4 (23,2) to (23,5)

Figure 9 shows the layout of the store, with the shelf fittings shaded in. The top left hand corner is position (1,1).

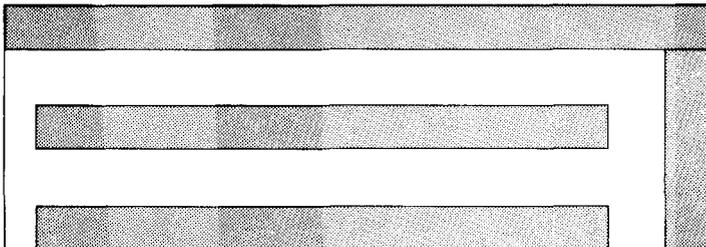


Fig. 9 A Store Layout

Operations from the Layout module are used to construct a representation of the floor plan and fittings. The types needed are Layout, Position and Shelf:

```
Layout = pair(Position,set(Shelf))
Position = pair(Nat,Nat)
Shelf = map(Position,Category)
```

The operations needed to construct a layout are:

```
emptylayout : Position → Layout
emptyshelf : Position × Position → Shelf
addshelf : Shelf × Layout → Layout
```

emptylayout takes an object of type Position, which is a vector specifying the dimensions of the floor plan – in this case, (23,5). emptyshelf constructs a shelf spanning the two positions supplied as parameters; it is empty in the sense that no products have been allocated to it. addshelf adds a shelf to an existing layout. The layout shown in Fig. 9 is constructed by:

```
sh1 == emptyshelf((1,1),(23,1)) ;
lay == addshelf(sh1,emptylayout((23,5))) ;
```

and so on for the other shelves. An operation to display a layout is also provided:

```
display : Store → ScreenDisplay
```

where the definition of display and the representation of ScreenDisplay are machine-dependent. Output from the display function is shown in Figs. 10, 11 and 12.

3.9 Constructing the Store

All the data objects which comprise the store have now been constructed. Recall from Section 2 that a store is represented by a record containing eight fields:

```
Store = record(inventory : Inventory,
               suppliers : Suppliers,
               orders : Orders,
               sales : Sales,
               products : Products,
               bartable : Bartable,
               customers : Customers,
               layout : Layout)
```

The store can now be built by applying the record constructor mk_store to the data objects constructed in Sections 3.1–3.8:

```
st == mk_store(invt,sup,ord,sal,prod,bar,cust,lay) ;
```

This data object will be used in the four applications which follow.

4.1 (a) Space Allocation

Given the range of products carried by the store, and the layout of the store and its shelving, the store management have to plan how much shelf space to allocate to each product and where to locate it. Related products are normally shelved together. Complementary products are sometimes located close together (e.g. sage and onion stuffing near the poultry). Location of perishables is also influenced by storage requirements – chilled, frozen, and so on. The amount of space devoted to a particular product will be affected by factors such as rate of movement, the size and sometimes the shape of the package and the frequency of deliveries. There are other constraints on space allocation such as the legal restriction on off-sales hours, meaning that alcohol must be shelved in such a way that it can be cordoned off from other goods outside off-sales periods. Other considerations include locating large and/or heavy packages on lower shelves, and hazardous products out of reach of children.

Algorithmic solutions to this class of problem are computationally expensive, since the space of possible solutions is very large (the combinatorial explosion problem). However some software aids for facilities management in general [7], design of layout [8] and space allocation in particular, (e.g. SPACEMAN, Accuspace, Resource-Opt [9]) are available. Expert system technology, using heuristics to guide the search, offers another possible approach [6].

In this application we simplify the problem by restricting ourselves to the task of shelving related products together. We have already seen that the product database is organised hierarchically; the problem then is to distribute the products according to their hierarchical relationships across the physical shelf space in the supermarket. Given any arbitrary layout, the space allocation algorithm will generate a plan locating products on shelves. The inventory database is used to determine total shelf space allocation per product, and the spatial arrangement of different products is determined on the basis of the information inherent in the hierarchical structure of the product database. So, for example, all coffee products should be located together, and, within that group, all the instant coffees should be together, and so on down through the levels of the product hierarchy.

The Space Allocation module imports the Layout module, so we can start by using the display operation to display the layout of the store. As yet, no products have been allocated to shelves.

```
display(st) ;
```

produces the output shown in Fig. 10.

instant coffee, brands, and different sizes of the same brand, are located adjacently.

The store manager can now inspect the proposed plan, and reallocate shelves if desired. The allocate operation takes a product name (of type Category), a position and a store, and returns a new store wherein the position is now occupied by the nominated product:

allocate : Category × Position × Store → Store

Suppose, for example, it is decided that Master Blend should be replaced by Irn Bru in position (1,1). The following reallocation would achieve this:

st2 = = allocate("irn bru 2l", (1,1), st1) ;

The result of this reallocation is shown in Fig. 12.

display(st2) ;

```

i m m m m m m m m m n n n n n n n n n n N N N
                                     N
   i i i f f f f f f f f f F F F F F F F   N
                                     N
   i i i i i i i c c c c c c c c c c N N N   N

m  master blend 230g
n  nescafe 50g
N  Nescafe 100g
c  cafe hag 50g
i  irn bru 2l
f  felix tuna 400g
F  Felix rabbit 400g

```

Fig. 12 Output from the display operation showing a Layout which has been altered manually

Further reallocations could be made. In this case, however, the manager decides that the original plan is satisfactory, and reverts to store st1.

4.2 (b) Automatic Perpetual Inventory

Goods are delivered to the store (goods inwards) and are stored in the store room before being put on the shelves. As customers' purchases (goods outwards) deplete shelf stock the shelves must be replenished from store room stocks. When the total stock level of a product falls to a certain level, stock is replenished by ordering more stock from the suppliers.

Traditional stock control procedures rely on periodic stock taking and reordering of goods from suppliers as and when required. Goods inwards figures are derived from delivery documentation and goods outwards is not measured directly, but inferred from the results of stock taking. The stock

level at which reordering takes place depends amongst other things on the rate of movement of the product, the expected delivery time, the shelf life of the product and the in-store storage capacity. A further complication is that rate of movement of certain products is subject to seasonal variation.

The introduction of electronic scanning equipment at checkouts makes it possible to measure goods outwards directly, and therefore to monitor stock continuously. Orders to suppliers can be generated automatically (teleordering). Accumulated scanning data can be used to generate management reports and predict demand. EPoS systems incorporating teleordering are already in use in many retail sectors, for example bookselling [10], the brewery trade [11], jewellery [12]. EPoS is also being taken up by smaller retailers: a recent study by RMDP showed that nearly half of the 450 small multiples surveyed had installed EPoS systems [13].

This application exploits scanning data in order to maintain a perpetual inventory system. At the checkout each item purchased is passed over an optical scanner which reads the barcode printed on the product. The price of each item is retrieved from the product database and accumulated to produce the customer's bill. At the same time the inventory database and sales are updated and stock levels are tracked so that orders to suppliers can be generated automatically. Since shelf stock levels are held in the inventory database, the need for shelf filling can be signalled before stock-outs occur.

Here the assumption is made that goods inwards equals goods outwards. (In reality the situation is complicated by "shrinkage", a euphemism for theft by staff and shoplifters.)

We start with the store in its initial state. If the orders and sales databases are inspected, it can be seen that as yet no sales have been made, and therefore no orders to suppliers have been accumulated.

```
orders(st1);
  {};
sales(st1);
  0;
```

Now we inspect the inventory records for two of the products. `showinv` requires a barcode rather than a product descriptor, so we use the barcode table to convert from a product descriptor to a barcode.

```
showinv(getbar("felix tuna 400g",bartable(st1)),st1);
  {"ONSHelves" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 15, "REORDLEV" -> 10 };
showinv(getbar("master blend 230g",bartable(st1)),st1);
  {"ONSHelves" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 15, "REORDLEV" -> 10 };
```

The inventory is in its initial state, with 10 units of shelf space allocated to

each product, the shelves full and 15 units of each product in the store room. So in total there are 25 units of each product in the store. Now a customer comes into the store and fills a basket with the following items:

<i>Item</i>	<i>Qty</i>	<i>Barcode</i>
Master Blend ground coffee (size 230g)	4	5000136997729
Nescafe instant coffee (size 50g)	3	5000243000204
Felix tuna cat food (size 400g)	5	5000108005445
Felix rabbit cat food (size 400g)	1	5000108005353

and presents them at the checkout. The items are passed over the scanner. To simulate this, we define a data object *bas1* to represent the scanning data for the customer's purchases. *bas1* is an object of type *Basket*, and as such is represented as a sequence of barcodes:

Basket = seq(*Barcode*)

bas1 is now defined:

```
bas1 = =
[5000136997729,5000243000204,5000108005445,5000108005353,
 5000108005445,5000136997729,5000108005445,5000136997729,
 5000136997729,5000243000204,5000108005445,5000108005445,
 5000243000204];
```

This represents the data which the barcode scanner would capture for this customer transaction. The transaction is performed by the operation *buy*:

buy : *Basket* × *Store* → *Store*

The *buy* operation returns a new store which has the inventory, sales and orders fields altered appropriately. The store resulting from the transaction is defined as *st2*:

st2 = = *buy*(*bas1*,*st1*) ;

For each of the products in the basket, the shelf stock recorded in the inventory is decremented by the number of units of that product purchased. If this results in the shelf stock falling below the level specified by *MIN-SHELF*, the shelf is refilled from the store room and the store room stock *INSTORE* is decremented. The effect on the inventory can be seen by inspecting entries for two of the products. (Changes to the inventory are shown here in boldface type.)

```
showinv(getbar("felix tuna 400g",bartable(st2)),st2) ;
{"ONSHELVES" → 10, "MINSHELF" → 5, "TOTSHELF" → 10,
 "INSTORE" → 10, "REORDLEV" → 10 } ;
showinv(getbar("master blend 230g",bartable(st2)),st2) ;
{"ONSHELVES" → 6, "MINSHELF" → 5, "TOTSHELF" → 10,
 "INSTORE" → 15, "REORDLEV" → 10 } ;
```

Five tins of Felix tuna cat food were bought; this resulted in the shelf stock falling to its minimum level, and consequently the shelf stock has been

replenished from store room stocks, which are now reduced to 10. In contrast, four packets of Master Blend were bought; the shelf stock has fallen to six, and the store room stocks remain as before.

We now turn to the orders. Every time items are purchased, an entry is made in the orders database. Orders are accumulated so that as many items are reordered as have been sold. Reordering then is tailored to the current rate of movement of goods. This represents a very simple demand model, where demand in the immediate future is assumed to match current demand. As the orders are built up, the cheapest supplier of each product is selected for the next order. When total stock of any product falls below its reorder level, all the orders in the system are despatched simultaneously. (This of course represents another simplification of the real situation). After the purchase of bas1, the orders database contains the following information:

```
orders(st2) ;
  {sc3 -> {5000108005445 -> 5, 5000108005353 -> 1},
   sc2 -> {5000243000204 -> 3},
   sc1 -> {5000136997729 -> 4}} ;
```

None of the products in the store has reached its reorder level, so the orders are not despatched yet. Inspection of the suppliers database confirms that each product is to be reordered from the (currently) cheapest supplier.

Meanwhile, the sales figure is incremented. The total cost of the transaction is calculated from the *sell* prices in the product database, and the value of sales is altered accordingly:

```
sales(st2) ;
  11.41 ;
```

Another customer arrives and purchases some goods. To simplify the explanation, this customer purchases exactly the same range of goods as the previous one.

```
st2 = = buy(bas1,st2) ;
```

Changes to the inventory of st2 can be illustrated by applying showinv again:

```
showinv(getbar("felix tuna 400g",bartable(st2)),st2) ;
  {"ONSHELVES" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 5, "REORDLEV" -> 10} ;
showinv(getbar("master blend 230g",bartable(st2)),st) ;
  {"ONSHELVES" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 7, "REORDLEV" -> 10} ;
```

This time shelf stocks of Master Blend have been replenished. If we look at the orders file, we see that the same suppliers have been chosen as previously, since no changes have been made to the suppliers' pricelists. Quantities of goods to be reordered again match quantities of goods sold.

```
orders(st2) ;
  {sc3 -> {5000108005445 -> 10, 5000108005353 -> 2},
   sc2 -> {5000243000204 -> 6},
   sc1 -> {5000136997729 -> 8} } ;
```

And, not surprisingly, sales have doubled.

```
sales(st2) ;
  22.82 ;
```

Yet another customer arrives at the checkout, and by coincidence buys the same selection of goods.

```
st2 = = buy(bas1,st2) ;
```

If we look at the orders database now

```
orders(st2) ;
  {} ;
```

we find that it is empty. This purchase resulted in stocks of Felix tuna cat food falling below the reorder level, so all the orders have been despatched. Time is not represented in this system, so the goods are delivered as soon as they are ordered, as we can see from the inventory.

```
showinv(getbar("felix tuna 400g",bartable(st2)),st2) ;
  {"ONSHelves" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 15, "REORDLEV" -> 10 } ;
showinv(getbar("master blend 230g",bartable (st2)),st2) ;
  {"ONSHelves" -> 6, "MINSHELF" -> 5, "TOTSHELF" -> 10,
   "INSTORE" -> 19, "REORDLEV" -> 10 } ;
```

Stocks of each product have been boosted by the number sold, and now total stock of each good in the store is back to the initial level of 25. Finally we look at the sales figure:

```
sales(st2) ;
  34.23 ;
```

and find that the day's takings so far amount to £34.23.

4.3 (c) In-Store Information

One Canadian chain has stores which are so large that customers frequently have difficulty finding the products they wish to buy. The company has installed terminals in the store so that customers can request directions from their present position to the location of a particular product. This is the purpose of this application. The algorithm is general in that it is not tailored to any specific layout.

For any object of type Layout, the where operation generates instructions to the customer to find a particular product:

```
where : Position x Category x Store -> Instructions
```

The first parameter specifies the customer's current location, the second names the product they wish to find. The representation of Instructions is

Instructions = seq(Move)

where

Move = pair(Direction, Nat)
 Direction = {"North", "South", "East", "West"}

Figure 13 shows the current layout of the store, with the position (11,2) marked X. A customer at this position wishes to buy a 50 gramme jar of Cafe Hag.

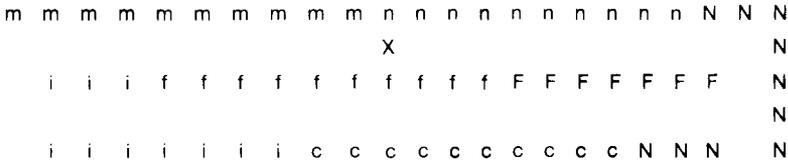


Fig. 13 Customer positioned at (11,2) trying to find nearest Cafe Hag (c)

The where operation will find all possible routes (ignoring cycles), select the shortest, and convert the route into intelligible instructions. (It also cleverly avoids directing customers through shelving.) The algorithm is based on a standard route-finding algorithm from graph theory, but is complicated by the fact that the desired product may be distributed across several different locations in the store. The nearest location is selected, and the shortest path to that location is found:

where((11,2), "cafe hag 50g", st2) ;
 [(East, 11), (South, 2), (West, 4), (South, 1)] ;

Of the two possible routes, the shorter one, starting East rather than West, is chosen.

4.4 (d) Teleshopping

Teleshopping is a shopping facility which enables shoppers to select and order goods over a communications network. This is an extension of the concept behind the interactive video disk used for point of sale promotions, such as Asda Stores' Videosystem system [14]. The Videodem system allows customers to browse through a product catalogue and view demonstrations of products in-store.

For teleshopping, an interactive video disk and/or videotex machine may be linked to a magnetic stripe reader, so that customers can complete a purchase by debit card or credit card transaction. Alternatively, the system may just register customer orders. The teleshopping terminals may be placed in-store, or in a public area such as a shopping mall, or in the customer's home. The American Comp-U-Card "Shopping Machine" system, in use in

the USA, is used amongst other things for selling holidays [15]. The Danish Call-Shop system began pilot testing in grocery outlets in Norway (for sale of non-food lines) in 1986 [14], and was introduced in the UK in 1988 [16]. The Bradford Centrepoint Project offers a free home shopping service to the elderly and disabled, who use the service mainly for food purchases [17]. Food shopping is also included in the Telecard Supershop system, based on the Prestel service in the UK, and in the French system based on the Teletel network [17].

In our teleshopping application it is assumed that suitable communications and EFT systems are in place, and that the customer has a terminal of some kind installed at home. The customer is given a certain level of (read only) access to the retailer's database, and may browse through the product information and order goods. The database can be regarded as an online catalogue.

In our food retailing example, the customer would be able to submit queries concerning products, such as:

“What kinds of coffee are sold in the store?”
“Is there any decaffeinated coffee?”
“Which is the cheapest instant coffee?”
“What would be a good substitute for product X?”
“Which foods contain (or do not contain) ingredient Y?”

The teleshopping application provides customers with seven operations: six query operations, and one operation for ordering goods:

show : Category × Store → Info
cheapest : Category × Store → set(Category)
cheapestprice : Category × Store → Nat
substitutes : Category × Store → set(Category)
containing : Category × Val × Store → set(Category)
notcontaining : Category × Val × Store → set(Category)
order : CustId × CustOrder × Store → Store

These types have been described earlier in connection with the Products and Customers modules, with the exception of CustOrder, which represents a customer's order for goods.

A customer might have the following interaction with the system. First they might work their way down from the top of the product hierarchy, to see which goods are available:

```
show("foods",st2)
  {beverages, soft drinks, pet foods} ;
```

The reader may notice that the user needs to know one of the categories (e.g. foods) in order to be able to start to use the system. This is not an important point; here we are interacting with the prototype. In an implementation, an interface such as a menu system would be provided.

```

show("beverages",st2) ;
  {coffee} ;
show("coffee",st2) ;
  {ground coffee, instant coffee} ;
show("instant coffee",st2) ;
  {nescafe, cafe hag} ;

```

Having seen which product groups are available, and having focused in on beverages and then on coffee, the customer is now interested in prices:

```

cheapest("coffee",st2) ;
  {nescafe 50g} ;
cheapestprice("coffee",st2) ;
  85 ;

```

The customer now wants to find out more about a tentatively selected product:

```

show("nescafe 50g",st2) ;
  {contains -> {coffee solids, caffeine},sell -> 85,size -> 50} ;

```

and, worried about caffeine, asks which beverages do not contain it:

```

notcontaining("beverages","caffeine",st2) ;
  {cafe hag 50g} ;

```

The customer then asks what brand substitutes for Nescafe are available:

```

substitutes("nescafe",st2) ;
  {cafe hag} ;

```

and what alternatives there are to ground coffee

```

substitutes("ground coffee",st2) ;
  {instant coffee} ;

```

Again feeling health-conscious, the customer asks which foods contain fibre and not sugar. (inter gives the intersection of the two sets.)

```

containing("foods","fibre",st2)
inter
notcontaining("foods","sugar",st2) ;
  {felix tuna 400g, Felix rabbit 400g} ;

```

and finds that, with this store's impoverished range, the cat has the best chance of a healthy lifestyle. Eventually the customer is ready to place an order. We simulate this in the prototype by defining a data object of type CustOrder, where

```

CustOrder = map(ProdDesc,Nat)

```

By another coincidence, this order corresponds exactly to the basket bas1 which was used earlier. However a more suitable representation is used for customer orders, since customers should not be required to deal with

barcodes, and specifying quantities is more convenient than repeating items.

```
co1 == {"master blend 230g" -> 4, "nescafe 50g" -> 3,  
        "felix tuna 400g" -> 5, "Felix rabbit 400g" -> 1};
```

The total cost of this order is £11.41. If the store manager were to inspect the inventory, orders and sales now, the following state of affairs would emerge:

```
showinv(getbar("master blend 230g",bartable(st2)),st2);  
{"ONSHELVES" -> 6, "MINSHELF" -> 5, "TOTSHELF" -> 10,  
 "INSTORE" -> 19, "REORDLEV" -> 10};  
orders(st2);  
{ };  
sales(st2);  
34.23;
```

Currently two customers are authorised to order goods using the electronic shopping service. They are recorded, along with their credit limits, in the customer database:

```
customers(st2);  
{morag -> 2000, mac -> 1000};
```

Mac attempts to order goods:

```
st3 == order("mac",co1,st2);
```

and fails, since his credit is not high enough. The inventory, orders and sales and customer databases therefore remain unchanged:

```
showinv(getbar("master blend 230g",bartable(st3)),st3);  
{"ONSHELVES" -> 6, "MINSHELF" -> 5, "TOTSHELF" -> 10,  
 "INSTORE" -> 19, "REORDLEV" -> 10};  
orders(st3);  
{ };  
sales(st3);  
34.23;
```

```
customers(st3);  
{Morag -> 2000, Mac -> 1000};  
st2 = st3;  
TRUE;
```

Now Morag attempts a similar transaction:

```
st3 == order("morag",co1,st2);
```

and is successful, as we can see from the new state of the store:

```
showinv(getbar("master blend 230g",bartable(st3)),st3);  
{"ONSHELVES" -> 10, "MINSHELF" -> 5, "TOTSHELF" -> 10,  
 "INSTORE" -> 11, "REORDLEV" -> 10};
```

```

orders(st3) ;
  {sc3 -> {5000108005445 -> 5, 5000108005353 -> 1},
   sc2 -> {5000243000204 -> 3},
   sc1 -> {5000136997729 -> 4}} ;
sales(st3) ;
  45.64 ;
customers(st3) ;
  {morag -> 859, mac -> 1000} ;

```

Morag's credit has been decremented, sales have been incremented, and the orders and inventory files have been updated to reflect that fact that her order is now being despatched.

Finally, it is worth noting that all the query operations from this application could also be incorporated into the in-store information application.

5 In Conclusion

The retail system was designed using the **me too** method of software design. This method combines the techniques of formal specification and rapid prototyping. The specification is modularised, and comprises 18 components. The method is iterative: the user first specifies an abstract model of the objects (types) and operations on those types, for the system being studied. In the next step representations are given to the objects, and definitions to the operations. Then the specification is typed into the **me too** shell and executed as a prototype. This provides feedback on the design, exposing errors and inconsistencies. The specifier returns to the previous step to correct any errors and possibly to explore alternative designs. The process is so rapid that many iterations may be made within a short time; the retail specification was the result of several iterations.

The full specification of the retail is too long to reproduce here in full; however samples are given in Appendix 1 and a copy of the complete specification can be obtained by writing to the author at the Department of Computing Science, University of Stirling.

Appendix 1

Extracts from the Formal Specification of the Retail System

As illustration, we give here the specifications for the store component and for two of the components which it imports, namely Inventory and Suppliers.

```

% _____
%                               SPECIFICATION of STORE
% _____
% IMPORTS
% Inventory, Suppliers, Orders, Sales, Products, Bartable, Customers, Layout

```

```

% -----
% MODEL
% OBJECTS
%   Store
% -----
% SPECIFY
% OBJECTS
%   Store = record(inventory : Inventory, suppliers : Suppliers, orders : Orders,
%                 sales : Sales, products : Products, bartable : Bartable,
%                 customers : Customers, layout : Layout)
% RECORD DECLARATIONS
%   Store : : inventory suppliers orders sales products bartable customers layout;
% -----
% -----
%                               SPECIFICATION of INVENTORY
% -----
% IMPORTS
%   Bag
% -----
% MODEL
% OBJECTS
%   Inventory – the inventory data for the store
%   Barcode  – 13-digit number uniquely identifying a product
%   InvEntry – data stored about each product in the inventory
%   Attr     – inventory attribute, such as number of items on shelves
% PUBLIC OPERATIONS
%   emptyinv : -> Inventory
%   mkinv : Barcode × InvEntry -> Inventory
%   updinvent : Barcode × Attr × Int × Inventory -> Inventory
%   getinv : Barcode × Attr × Inventory -> Nat
%   joininv : Inventory × Inventory -> Inventory
%   mkent : Nat × Nat × Nat × Nat × Nat -> InvEntry
% emptyinv constructs the inventory with no entries
% mkinv constructs an inventory with one entry
% updinvent returns an inventory with a single entry, where the value of attr is
%   incremented or decremented (depending on the sign of Int) for the product
%   indicated by barcode
% getinv retrieves the value of an inventory attribute for a product
% joininv merges two inventories
% mkent constructs an inventory entry
% -----
% SPECIFY

```

```

% OBJECTS
% Inventory = map(Barcode,InvEntry)
% Barcode = Nat
% InvEntry = bag(Attr) i.e. map(Attr,Nat)
% Attr = {"ONSHELVES","MINSHELF","TOTSHELF","INSTORE","REORDLEV"}

% OPERATIONS
emptyinv() == {};
mkinv(b,f) == {b -> f};
updinv(b,a,q,i) == mkinv(b,bagunion({a -> q},i[b,{}]));
getinv(b,a,i) == i[b,{}][a,{}];
joininv(i1,i2) == i1 overwr i2;
mkent(on,min,tot,ins,re) ==
  {"ONSHELVES" -> on,"MINSHELF" -> min,"TOTSHELF" -> tot,"INSTORE" -> ins,
   "REORDLEV" -> re};
% -----

% -----
% SPECIFICATION of SUPPLIERS
% -----

% IMPORTS
% Set_Extensions, Numbers_Extensions
% -----

% MODEL

% OBJECTS
% Suppliers -data on potential suppliers of goods to the store
% Pricelist -a suppliers's price list (i.e. the store's COST price)
% Price -price in pence
% Barcode -13-digit number uniquely identifying a product
% Suppliercode, Name-Address - supplier's code and name and address

% PUBLIC OPERATIONS
% emptypricelist : -> Pricelist
% addprice : Barcode x Price x Pricelist -> Pricelist
% updprice : Barcode x Price x Pricelist -> Pricelist
% emptysuppliers : -> Suppliers
% addsupplier : Suppliercode x Name-Address x Pricelist x Suppliers -> Suppliers
% cheapestsup : Barcode x Suppliers -> Suppliercode

% PRIVATE OPERATIONS
% getprice : Barcode x Pricelist -> Price
% getpricelist : Suppliercode x Suppliers -> Pricelist
% cheapestcost : Barcode x Suppliers -> Price
% -----

% SPECIFY

% OBJECTS

```

```

% Suppliers = map(Suppliercode,pair(Name-Address,Pricelist))
% Pricelist = map(Barcode,Price)
% Price,Barcode = Nat
% Suppliercode,Name-Address = String
% OPERATIONS
emptypricelist() = = {};
addprice(b,p,pl) = = pl overwr {b -> p};
updprice(b,p,pl) = = pl overwr {b -> getprice(b,pl) + p};
emptysuppliers() = = {};
addsuppliers(sc,n,pl,s) = = s overwr {sc -> (n,pl)};
cheapestsup(b,s) = =
    any({sc|sc <- dom(s); getprice(b,getpricelist(sc,s)) = cheapestcost(b,s)});
getprice(b,pl) = = pl[b,0];
getpricelist(sc,s) = = second(s[sc,("Err",{})]);
cheapestcost(b,s) = =
    dmin({getprice(b,getpricelist(sc,s))|sc <- dom(s);
        b member dom(getpricelist(sc,s))});
% -----

```

Bibliography

me too

- [1] HENDERSON, P. *me too*: A Language for Software Specification and Model Building (preliminary report), *Internal Report FPN-9*, Dept. of Comp. Sci. Univ. of Stirling, Scotland, 1985.
- [2] HENDERSON, P. Functional Programming, Formal Specification and Rapid Prototyping, *IEEE Trans. on Software Engineering*, 1986, SE-12, 2, 241–250.
- [3] HENDERSON, P. & MINKOWITZ, C. The *me too* Method of Software Design, *ICL Technical Journal*, 1986, 5, 1, 64–95.
- [4] ALEXANDER, H. & JONES, V. *Software Design and Prototyping using me too*. London: Prentice-Hall International, 1989.
- [5] BENNETT, S., JONES, V., MINKOWITZ, C. & ROWLES, J. *me too Reference Manual*, Version 6.0, Tech. Rep. SETC/IN/209, STC Technology plc, Newcastle Under Lyme, UK, 1989.

Retail

- [6] JONES, V. & DAVIES, K. A Taxonomy of Application Areas for Expert Systems in Business, *Tech. Rep. TR-31*, Dept. of Comp. Sci., Univ. of Stirling, Scotland, 1986.
- [7] Keeping facilities under control, *Retail Automation*, Sept./Oct. 1988, 8, 5, p. 6.
- [8] Retail Europe – applying the technology, *Retail Automation*, July/Aug. 1986, 6, 4, pp. 8–9.
- [9] WALTON, P. Gut feeling for the right decision, *Data Link*, Oct. 14, 1985.
- [10] EPoS – going by the book, *Retail Automation*, July/Aug. 1986, 6, 4, pp. 4–5.
- [11] Allied-Lyons spends £8m on EPoS, *Retail Automation*, Jan./Feb. 1987, 7, 1, p. 7.
- [12] Focusing on the profit factor, *Retail Automation*, Jan./Feb. 1987, 7, 1, p. 9.

- [13] Smaller firms opt for EPoS, *Retail Automation*, Sept./Oct. 1988, 8, 5, p. 24.
- [14] Teleshopping – moving closer? *Retail Automation*, July/Aug. 1986, 6, 4, p. 3.
- [15] Computer help for Comp-U-Card, *Retail Automation*, Jan./Feb. 1987, 7, 1, p. 8.
- [16] Call Shop arrives in the UK, *Retail Automation*, Sept./Oct. 1988, 8, 5, pp. 19–20.
- [17] Teleshopping – this year, next year, sometime ...?, *Retail Automation*, Jan./Feb. 1987, 7, 1, pp. 13–14.

. . . towards a Geographic Information System

J. M. P. Quinn

FlagShip Project, MSTC, ICL West Gorton

Abstract

This paper describes the techniques developed in deriving a spatial model of a mapped area from a feature coded digital map of that area. It describes how the model is used in conjunction with a database application PLANES which allows users, such as Public Utilities and Local Authorities, to set up and interrogate large Geographic Information Systems.

Introduction

A Geographic Information System is an example of the extension of a conventional database to include spatial data such as that provided by digital maps. The spatial data can be used simply to provide a background against which the conventional data can be viewed or it can be used to extend the range of questions that the Information System is able to deal with to include spatial matters, such as: what road is behind a given house, which houses are within 100 metres of a given point, road or area? Which roads are affected by a given development? How does one get to ...? Show the houses served by a given distribution network or having a specific attribute.

The spatial data involved usually starts as an image either on paper or video. For those applications in which the textural information in the image is important (such as vegetation cover) it may well be appropriate to retain the image form, but for many situations it is the shapes within the image and their interrelations that are important and the extraction of relevant features will of necessity produce a simplified vector-based view. The vector form is the one used in the application described later for which the spatial data is provided by the digital maps available from the Ordnance Survey, though other companies are able to digitise from paper maps.

The display and manipulation facilities inherent in the use of Geographic Information Systems draw heavily on a number of features that until recently have been associated with expensive graphics work stations. This stimulated the development of an experimental mainframe-based graphics system and

the use of the relatively dumb terminals which have been used in this study. One advantage of this approach has been that the graphical manipulation facilities can operate directly on data held in the large databases that are still very much the province of mainframes and, because it is centrally held, can be accessed by many users in different ways and for different purposes. The fact that the cost of graphics work stations has dropped significantly does not invalidate the choice; it allows much better user interfaces to be provided, while maintaining the database in a central location.

Geographic Information Systems have a number of potential applications, from thematic displays of information such as that derived from Census data to their use in the Local Authority and Public Utility areas, in which much of their data is spatial and currently held in the form of paper maps and drawings. The investment necessary because of the huge amount of data involved means it will be some time before they are in general use.

Digital maps are one source of spatial data. Others have grown out of the Computer Aided Design area which allows the user to generate and manipulate spatial information either to provide digital versions of the original style of technical drawings or more effectively to derive such drawings from the three dimensional modelling systems that are becoming an accepted part of Computer Aided Design and Manufacture. Either way the drawings consist of a set of user-definable shapes linked into a hierarchy of shapes that can be manipulated flexibly and easily.

Images from either satellite or high altitude flying provide yet another source of spatial data with a resolution which is rapidly approaching that possible with the use of conventional surveying techniques. The software and hardware technology required to convert such data so that it can be effectively used is already available from a number of companies but is still too expensive for widespread use. It can, for example, cost more than £500 to convert a 500 metre square paper map image into the equivalent vector form. The development of expert systems and parallel processing as well as conventional systems should help matters over the next few years.

There are several developments both inside and outside ICL that are moving towards an effective Geographic Information System. These are:

- (a) PLANES, a Planning, Land, Network and Spatial system based on an IDMS database and TP system which is described later.
- (b) the availability of large scale (1:1250) scale maps, or the technology to produce them, covering significant portions of the country.
- (c) MapGin, a system which allows spatial objects to be identified from large scale digital map data, stored and manipulated.
- (d) the software and communications facilities offered by ICL's VME operating system which allow a variety of graphics terminals to be networked with a mainframe computer which can itself be distributed.
- (e) the development of expert systems supported by declarative languages

and parallel processing; this will allow further expansion of the current trends.

In this paper the MapGin system and its rationale are described, as well as some aspects of the PLANES system with which it is being used, as one example in which spatial information can be included effectively.

MapGin Background

MapGin is a mainframe-based system that is primarily concerned with the interpretation of digital maps of the type produced by Ordnance Survey; and started life as a way of displaying such maps at arbitrary scale on a variety of graphics terminals.

A major impetus for this work came from the realization that a large amount of information is held implicitly in digital maps that can normally only be understood and used by humans. The maps are held in digital form for convenience and used for the draughting of paper maps which can be used to provide a backdrop for service distribution networks, for example gas or electricity, or any form of distributed information. While these are sensible uses for the maps there is potential for much more and this paper describes the start of an exploration of such possibilities.

As mentioned earlier the two basic forms of storage for spatial data are, as an image, or as a set of simplified shapes delineated by lines or vectors. Each form has different properties and uses and one can move from one to the other in the way illustrated by Fig. 1. In this an object is shown to be composed of both logical and spatial information.

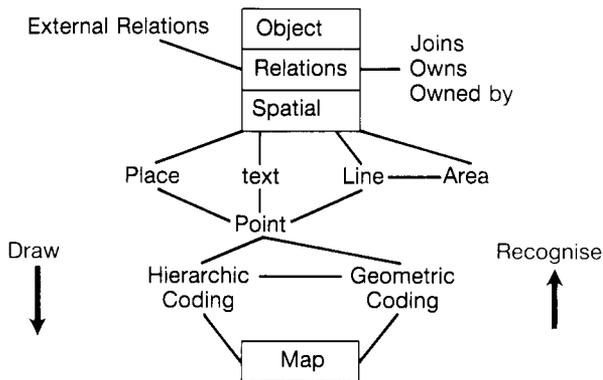


Fig. 1 The relationship between an Object and image view

The logical information relates an object to other objects while the spatial information defines the shape and position of the object in terms of primitive components: points, lines and areas. Any human readable information such as text also needs to be included. The components may also be defined in

terms of sets of simpler components, for example an area in terms of its boundary lines. This produces advantages in terms of consistency of detail, and reduced volume of data.

The act of drawing the map moves the focus down from the objects through to their spatial components. From there each element is reduced to a set of points in the displayed image. The act of recognition implies moving the focus the other way, first grouping points into features then features into objects. With OS maps one is able to start the recognition process at the feature level, thus enabling a concentration on the process of object identification and handling.

During the development the methods of transforming the feature-coded image into an object-based view have been explored. The techniques investigated are based on using similar visual clues to those that enable a human observer to make sense of the patterns of coded lines that form the map. By employing these techniques it has been possible to use the maps in a far wider arena than previously.

In MapGin the original map data is retained and used for displaying areas of interest in the map while the derived objects are used to make sense of that view.

The main features of the Ordnance Survey data investigated are:

- (a) The volume of data is large.
- (b) the data is often geometrically imprecise.
- (c) the spatial algorithms needed to extract meaning out of the data are highly dependent on the nature of the data.
- (d) the data appears to be coded in different styles with varying degrees of precision. In some cases the data has sections missing and is inconsistent with itself.
- (e) the cartographic data is provided in National Grid Coordinates and thus requires two six digit numbers to be manipulated.

These considerations led to an initial design philosophy for the system such that:

- i. the most useful key to large scale urban data is the road and street network, both logically and geographically. This later provided a convenient link into the PLANES system.
- ii. all algorithms should allow a degree of tolerance to inaccuracy.
- iii. while investigating an area the objects set up should be temporary, created as needed during a given session. Once the analysis appears complete it should be possible to save the objects in a database from which they can be quickly restored when required.
- iv. major discontinuities in source data will be corrected by visual interpretation and manual intervention. The system should attempt to identify such situations and allow correction. Minor inaccuracies, such

as lines that don't join exactly, can safely be interpreted and if appropriate corrected.

- v. the means of interacting with the system should be simple, easy to use and extensible.

This approach places requirements on the supporting system such as:

- (a) the hardware system should provide a large store to accommodate the large data set.
- (b) a full general purpose graphics package is required to handle the spatial data and perform the necessary transformations to the original data and allow it to be displayed on as wide a set of terminals as possible.
- (c) fast floating point manipulation, because the data size is large and many geometrical calculations are involved.
- (d) powerful multiprocess and multiuser facilities are also needed to facilitate independent and cooperative operation.
- (e) flexible communication facilities to allow networking of the various types of terminals.
- (f) the software should be implemented in a high level language that provides flexible object handling, access to the system facilities and mixed language capabilities.

The Spatial Model

The spatial model adopted is based on the road network of the area. It consists of sets of objects of various types. The objects are represented by data structures that include their logical, textual and spatial attributes. The main object types involved are defined for the Roads, Road Junctions, Properties and distribution networks.

The data on which the model is based is provided by various Ordnance Survey digital maps at scales of 1:1250 and 1:2500 and is accurate to 0.5 metres. One set of maps covers part of Coventry. The set covers an area of 72 square Kilometres in 110 map sheets. Metropolitan areas account for 84 500 metres square map sheets and country areas for 26 1 Kilometre square map sheets.

The data is held on a map sheet basis and consists of sets of over 100 feature types which can be lines (represented as lists of coordinates), text or point coordinates. These features are spatially, not logically, associated and each feature type is accompanied by qualifying information. In the case of lines and points the feature code and line style for drafting is included. For text the location and orientation are provided together with the text string itself.

The algorithms developed for extracting the objects from the feature coded map data can be viewed as a set of empirical rules which are specific to this style of feature coding. If the base representation is changed then the rules must be changed also. At the present time we have chosen to group the

objects within map sheets in the same way as the original feature coded data, although this is not a fundamental obstacle.

Identification of Objects

The information held for the two main object types is basically the same: there is a constant part to identify and categorize the object and link it with other ones, a variable set of coordinate points to delineate it and a containing rectangle used for fast location of an object.

A road is identified as a set of centre line vectors, of a specific road centre line feature code with a single associated name; and a house is a closed polygon of coordinates also of a given feature code type. The header information for a road includes the road name and type while that for a house includes its name, type and the set of OS numbers associated with it in the case of flats or terraced housing. Terraced and semi-detached houses can be further subdivided using the fence/property boundary feature codes.

The strategy for the allocation of names to roads is:

a road is a set of vectors with a single name within 2 metres of the line of vectors and an orientation parallel to those vectors

or

a road is the set of centre line vectors with no name
and is either continuous with a named road whose name it can take
or joins or is joined by a named road whose name it can take

or

a road is the set of centre line vectors with multiple names associated with it along its length
and can split up the original road at either junctions with other roads*
or bends so that each section of the original road has a unique name.

* In the case where there are several junctions between two names then one is chosen by looking at house numbers and choosing the junction nearest the smallest number. This technique starts with the apparent Ordnance Survey convention of orientating house numbers to lie parallel to the roads in which they are located.

After the first pass about 60% of the roads have been named uniquely. The final success rate is over 90%.

The strategy for identifying properties is based on the definition of a property as a closed polygon bounded by lines of one of the OS building feature codes. The area defined by the polygon will usually include either a number or a name text string. It will also have an associated road within 30 metres in a direction indicated by the orientation of the text.

The next two figures are direct screen dumps. Figure 2 shows the property polygons, including the numbers. The dividing walls are also shown together with the other property boundaries, the road edge is shown as a chained line.

The cursor is placed over the house of interest and the corresponding address displayed as 17 Dysart Close.

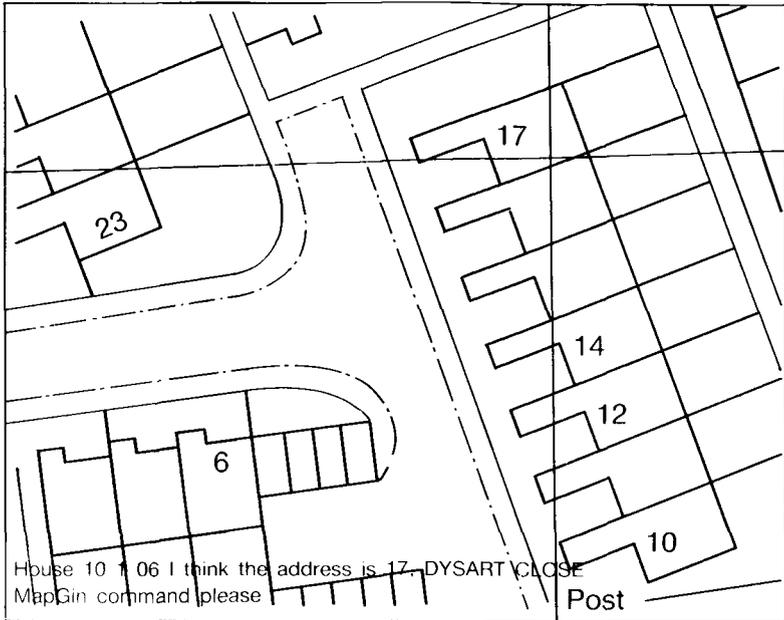


Fig. 2. Selecting a property to determine its address

The geometry involved in attaching the properties to the roads is shown in Fig. 3. The containing boxes for the properties are shown in black. Also shown are the road centre lines and the construction lines dropped onto the nearest roads as perpendiculars from the centre of the containing rectangle. If that is not possible then the nearest point on the road is taken. The road chosen is then the closest one in the direction of the numbers.

In properties, such as terraced housing, which include multiple numbers within the enclosing polygon the boundary-fence feature codes are used to split up the property polygon into house polygons each with its own associated number or name. The technique used in splitting up the outline is to transverse that outline in a clockwise direction turning clockwise at each intersection with the property boundary, thus isolating each house as a closed polygon within the overall enclosing polygon.

Figure 4 shows a terraced house that has been split up in this way. The calculated intersection points are shown as small rectangles, the house number included as text in the OS map. This represents an intermediate phase in the interpolation of house number. The final result is shown in Fig. 6.



Fig. 3. Determining associated road for a given property and showing the geometry involved

It is necessary to interpolate between the OS house number data in order to allocate missing numbers to the individual houses. However, the vagaries of the system of assigning numbers to properties precludes this in every case and some require user intervention.

If any current property crosses a map sheet boundary then the polygon is currently closed along the boundary. This is done for convenience since the original map data is organised in that way but is not a real problem provided both sides of the map edge match.

Database structures

There are three separate databases involved in the experimental system: the original map database and the object database which hold the spatial data, and the PLANES database which holds other textual attribute data.

Because of the nature of spatial data the map and object databases have been developed specially for variable length records and fast access. They are based on simple block files on to which a record structure has been superimposed which reflects the abstract data types used both for the map and object data. The position in the file of any map feature code or object

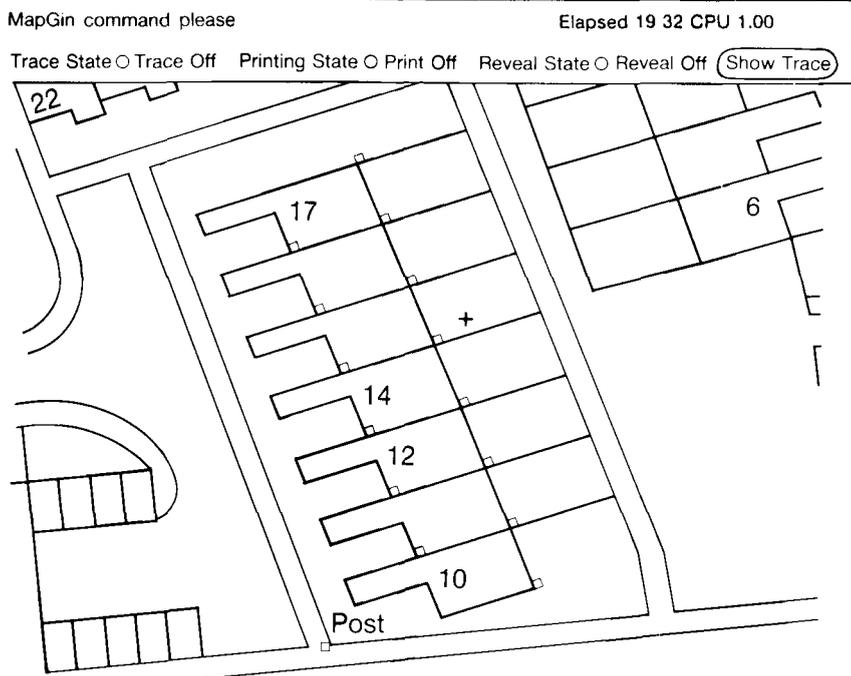


Fig. 4 Splitting up a terraced outline using the property boundaries

record is given by a File Address: the block number and offset the in block at which the item starts.

The map and the object database are organised on a map sheet basis. The map database has an associated index file that gives the northing and easting coordinates for each map sheet as well as the map width in metres, together with the file address of the start of its data. This allows the display of any part of the mapped area by selecting the map sheets which fall within the area of interest and then clipping the appropriate map sheet data to the screen.

The data for each map sheet is prefaced by a feature index that indicates which codes are included and the file address at which they start. The line data itself is held as a sequence of coordinate pairs, relative to the map sheet base, with a count of the number of points at the start of each sequence.

The map database allows records which span over more than one data block, the object database does not. This is because the map data is not intended for local modification but by updates from the Ordnance Survey. The object data is intended to be modified by improving the object extraction algorithm when a weakness in its operation is detected.

Experimental Spatial System

The main parts of the experimental system are shown in Fig. 5. It consists of three main subsystems that can be run independently to provide a subset of the total facilities, or together so that they can cooperate to provide the total set.

The subsystems are: MapGin which handles all the cartographic data, PLANES which handles the main property attributes and MapService which intercepts messages from either PLANES or MapGin and routes them to the appropriate recipient dependent on the context.

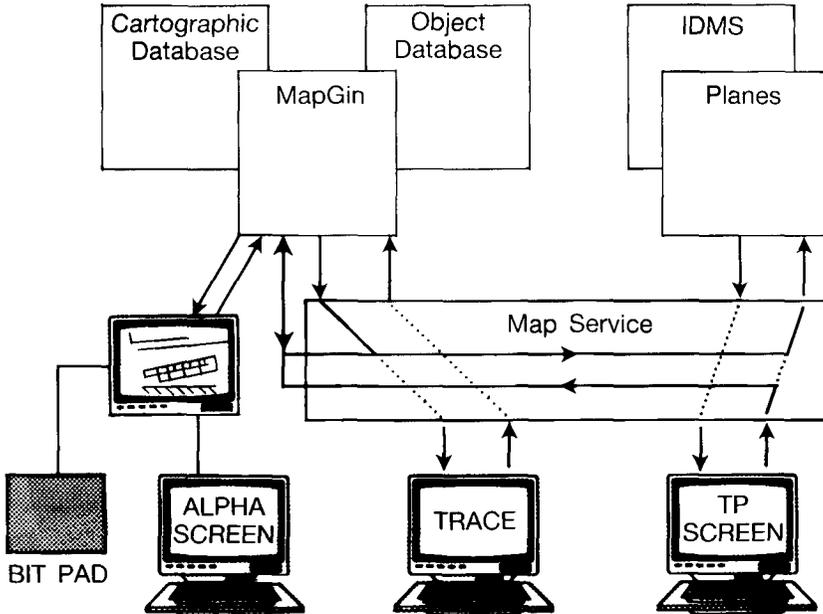


Fig. 5 Block diagram of the Experimental Spatial System

The lines show the interconnections and routes used. Each of the subsystems has its own terminal for trace information and MapGin also has a SUN3 graphics terminal with a mouse for cursor control.

There can be more than one MapGin subsystem connected. Each can either interact separately with PLANES or act as a slave to another MapGin subsystem. Both PLANES and MapGin can operate independently and need only be connected through MapService when either wants data from the other.

The system described runs on an ICL mainframe with the VME operating system (Virtual Machine Environment) capable of supporting multiple users.

A software environment has been developed to provide all the necessary graphical and communications facilities required. The basic graphical facilities are device independent and can be mapped on to various graphical devices in order to display and interact with the maps. Such devices are currently connected asynchronously either via Modems at up to 19200 baud or over OSLAN.

The set of devices that are supported are the Tektronix colour terminals 4501, 4701 and 4695 (printer), the Topaz greyscale graphics terminal, SUN 3, ICL 4602G terminal and DRS/PWS and DRS Model 30 ... It is simple to add further device drives.

Using MapGin experimentally

The user is able to interact with MapGin through a simple command language. Any positional information required is provided by using a cursor. The type of interaction is controlled by MapGin.

The user can select an area of interest and may move interactively about the mapped area, changing the scale as desired. It is also possible to move using the text feature included in the map data as a gazetteer. All positional information is held and presented in National Grid Coordinates.

In a given mapsheet the roads and junction objects can be determined by using the command WALK and the property objects by the use of the command PROPERTY. Once these have been used the objects can be saved using the SAVE OBJECTS command or interrogated using the STREET command for the roads or the ADDRESS command for the properties. In the latter command the graphics cursor is used to point to the road or property of interest.

On the SUN 3 and the PWS systems the user interface is presented as a set of pop up menus, text screens and windows. See Fig. 6 for a typical screen dump. The PLANES window is displayed in the top left hand corner showing data associated with 1 Dysart Close. Pop up menus are shown in the bottom left with a command selected. The main window displays Dysart Close which has been split into individual properties using the commands below the window.

An appropriate sequence of commands with explanation in { } would be as follows:

map 39	{display map 39, a 500 m sq map with origin at coordinates 434500, 279500}
walk	{identifies roads and junctions}
property	{identifies all the properties and allocate to roads}
save 39	{writes objects to a database}
mz	{zooms into a particular area indicated using the cursor}

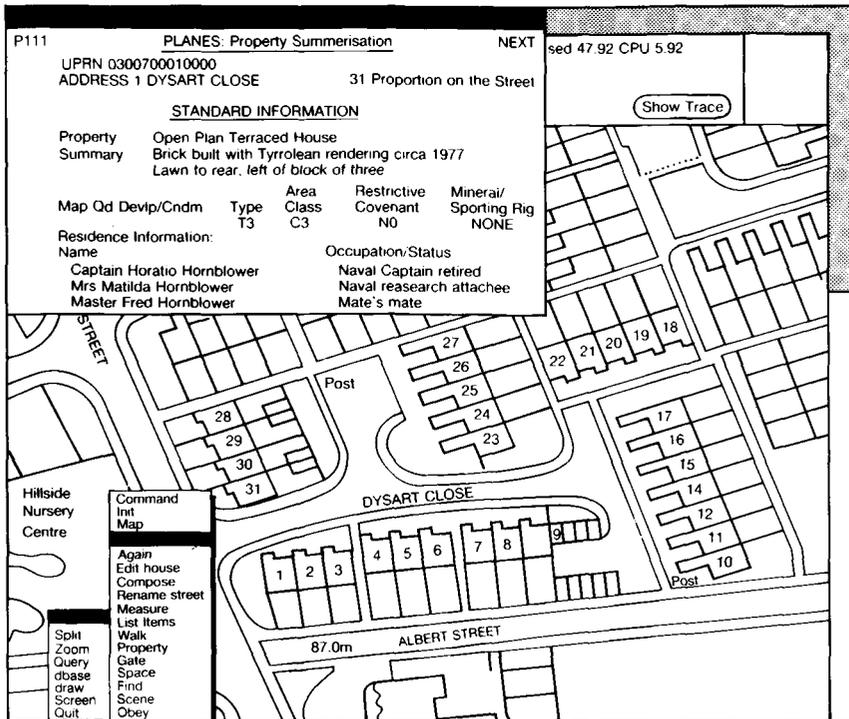


Fig. 6 A screen dump of a typical MapGin session with PLANES

connect {connect to the PLANES system using MapService}
 enquire {point to a building and display its address}
 street {point to a street to determine its name}
 split dysart {split Dysart Close into individual houses and interpolate the numbers}
 ... {continue with the interaction}

PLANES

PLANES is an ICL application being developed over a number of years to be the company's strategic Geographic Information System. Although its development is a long term programme the incremental and modular approach taken means that some components of the product are already in use at about 70 customer sites. Despite being still in the early stages, the distribution of these sites gives some indication of the wide potential application of these types of system – currently the major users are local authorities but also include public utilities and central government, with pilot projects underway with financial, retailing and commercial organisations. As for its geographic distribution, the system is already being used in the United Kingdom, Australia, New Zealand and Papua New Guinea and is gradually being introduced to further countries.

The system runs on an ICL mainframe running VME using IDMSX and TPMS, to provide the database handling and data processing.

There are conceptually four elements to the system, although they integrate closely to form the total Geographic Information System:

- (a) The Property Directory provides the ability to uniquely identify all properties and streets in the geographic area of interest by mapping an address on to a structured key. The Directory has extensive facilities to relate other data sets to this key, thus forming previously unrelated property-based applications into a corporate information system. It also provides a sophisticated search process, which may be used throughout this corporate system as a common access path to retrieve any of this data through the input of possibly incorrect or incomplete addresses.
- (b) The Network Directory performs the same job of providing a complete and unambiguous definition of a network, and acts as the focal point for network related applications and data sets. The networks described may be either highway networks or public utility ones such as electrical supply, sewerage etc. Network Directory has extensive facilities to define the elements of the network and the relationships between them in the most appropriate way. It can also support a time series view of a network, and map differing 'user' views onto the same underlying network definition.
- (c) The Spatial Directory holds mapping and spatial data in a form which allows it to be presented in the traditional manner of maps and plans, and to be classified as a set of objects to which textual data can be attached and between which relationships can be defined, in order to support a wide variety of analysis and decision support functions.
- (d) The Information Manager module provides a very flexible end-user system building capability which provides a simple way of generating applications which have inbuilt relationships with some or all of the above Directories. This provides a rapid way of generating some of the textually based application systems which are required to support an organisation's operational needs, complementing the Directories' ability to integrate existing applications.

Whilst the Property Directory, Network Directory and Information Manager are released products, as described above, the Spatial Directory currently exists only in prototype form. One of the major reasons for this module being the last to be tackled in our phased development is the difficulty and cost of collecting the spatial information in the form of structured objects as opposed to mere graphic images. While many users are very excited about the use of digital maps, both as means to access and view attribute information, and perhaps more importantly for the powerful analysis and data manipulation that becomes possible when spatial and a wide range of other attribute information are inter-related in a structured way, very few could afford the cost of the very labour-intensive data collection exercise that either digitising paper maps or surveying the bricks and mortar would entail. Thus the prospect of a piece of software such as

MapGin which could identify a high proportion of spatial objects automatically (and the remainder through some user interaction), both in terms of their spatial structure and their proper relationship to textual attributes such as property address or pipe number, would be a huge step forward in making Geographic Information Systems accessible to a large group of users who currently have the desire, or in many cases real need, for such a system but lack the resources to implement one.

MapService

This is an experimental program that uses VME communications facilities to connect the various components of the system together. It intercepts all the messages between PLANES and its TP terminal and can reroute all or a specific set of messages to MapGin. It can support the broadcast of messages to all the MapGin processes connected or handle them individually. As the development proceeds MapService facilities will be subsumed into the main PLANES system.

Using the Spatial Model

The interaction between PLANES and MapGin using the experimental facilities can be initiated from either end. Examples of such interactions are as follows:

- (a) MapGin can be used to display the area of interest and once the analysis has been carried out road names and property addresses retrieved by just pointing at the displayed image of the road or property. When connected to PLANES the address generated can be used to extract and display any associated attribute data: for example property descriptions, owners names and occupations.
- (b) The PLANES user may browse through a set of properties one by one and for any one can input the command VIEW. This command is intercepted by MapService and routed to MapGin so that the property can be displayed on the screen. In order to do that MapGin causes the PLANES screen to be redisplayed and a copy sent to MapGin. From this the position of the property can be determined and it is displayed at a scale that shows an area of 100 metres square centred on the property.
- (c) The PLANES user can select a property in a road and then WALK from one property set to another. At any stage he can request a walk to the next road, left, right or straight on. This command is routed to MapGin in order to determine the current road and property. The next road is selected on the basis of the nearest junction and then the first house chosen that satisfies the direction (left, right or straight on). This address is then passed to PLANES so that the enquiry can continue from the new property.
- (d) It is possible to use MapGin to determine and display the distribution of attribute data stored by PLANES within a given area. It is first necessary to select the shape of the area to be examined. This may be a

circle, a rectangle or an irregular polygon. Commands are then sent by MapGin to PLANES that cause it to walk through the set of properties with that attribute data and route the resultant PLANES data to MapGin. MapGin then checks whether the property is within the area. If it is the property is highlighted otherwise it is discarded.

In Conclusion

The MapGin experimental system has been used to demonstrate the possibility of constructing an object-based model of a feature-coded area with an acceptable degree of success. It is then possible to use that model to provide a graphical interface to an existing object based enquiry system such as PLANES such that the two parts of the system complement each other and the whole is greater than the sum of the parts.

The early experiment is being taken further and ways of incorporating the ideas of MapGin more directly into the PLANES system are being actively pursued. One immediate use is aimed at helping a new PLANES user to generate the structured key that is used to link the property address with the appropriate records in the IDMS database. This can be very time consuming and the facility of generating addresses from the OS map data should be very useful. Other uses are concerned with extending the analysis of map data to allow for the identification of land parcels, the introduction of symbol facilities and allowing the various distribution networks of Electricity, Gas and Water to be incorporated and displayed at various levels of abstraction.

Acknowledgements

I would like to thank Dr. I. Moshkun of ICL whose help and encouragement made the going easier. Mr. N. Perry of ICL Knowledge Base Systems whose initial work laid the foundations for this work. Mr J. M. Pratt currently with ICL in Reading for his help during the early stages of the project. Mr. C. J. Skelton, Project Manager of FlagShip for his support and understanding throughout the project, and to the members of the PLANES team for funding part of the experiments and mending their part when broken and other colleagues too numerous to mention

References

- UK Committee of Enquiry into the Handling of Geographic Information. Page 153. 'Handling Geographic Information' London HMSO 1987.
- Deriving and using an Object Based Model of a mapped area from a feature coded Representation. J. M. P. Quinn Vol 1 proceedings of Auto Carto London 1986. 59-68.
- Automatic Structuring and Feature Recognition. M. de Simone Vol 1 proceedings of Auto Carto London 1986. 86-95.
- The VME based Graphics Software System. N. Perry, I. Moshkun and J. M. P. Quinn FGIN 50 (unpublished).

“Ingres Physical Design Adviser”: a prototype system for advising on the physical design of an Ingres relational database

Michael Gunner

ICL (UK) Central Government Business Unit, Reading

Abstract

This paper describes a prototype advice system to help with the process of physical design of an Ingres relational database. The approach taken, and the design principles used, followed an investigation into ways of automating this process.

The prototype has rules for calculating query costs in terms of disk accesses. It does a search of alternative designs, comparing them, and advises which one is the best. This search is “heuristic” in that “rule of thumb” decisions are made to choose a few likely alternatives that are worth comparing, and it proceeds in stages which successively approach a solution.

1 Introduction

Ingres has been adopted as ICL’s strategic relational database system and there is therefore a major need for users and consultants to become acquainted with it. One of the things which is likely to receive most attention is physical database design, because of its importance in attaining good performance, and because it requires skills and experience which are scarce.

This paper describes an investigation into the use of design automation systems to help with relational database physical design. An expert systems approach was adopted in developing an inference mechanism and set of rules. This resulted in the production of a prototype advice system called ‘Ingres Physical Design Adviser’ (IPDA), written in Prolog and running on a personal computer (Prolog2 produced by Expert Systems International).

Section 2 discusses some related research described in published papers.

Section 3 is a description of the physical design process, and of what is involved in optimising performance.

Section 4 explains briefly the approach taken in designing IPDA.

Section 5 describes the design and general principles of IPDA. It includes an outline and a functional description. The main components in the prototype design are described.

Section 6 reviews the project as a whole: what was achieved and what possibilities there are, considering the implications of research papers reviewed in Section 2.

2 State of the art review and a survey of some published papers

2.1 Introduction

This section is a review of some published papers on performance and physical database design: different approaches to design aids, and the strategies adopted for solving design problems.

In summary this review considers papers and reports on the following subjects:

- the implications of an audited benchmark test,
- the place of performance calculations in possible Structured Systems Analysis and Design Method (SSADM) developments,
- a system which compares alternative sets of indexes using some broad measures of cost,
- a system called Design-by-Example which offers a user interface based on skeleton queries, as well as an interesting algorithm for searching for solutions. Again it is only for indexes and not storage forms,
- a system to generate representative query sets automatically from the logical design, based on what are seen to be the “general semantics” in the database,
- an Expert System for logical design of databases, part of a program to build expert systems for the complete database design process,
- an IBM system for selecting indexes which uses the database management system’s own optimiser to make the cost comparisons.

2.2 Ingres Silver Bullet Benchmark

The Ingres Silver Bullet Benchmark Report, audited by Codd and Date’s Consulting Group (Silver Bullet, 1988), claims to be a major milestone in the relational database marketplace. It reports a performance rate of 100 Debit/Credit transactions per second, achieved with a Sequent computer system with 16 processors.

Although achieving this level of performance helps in overcoming the barriers to the acceptability of relational databases for large-scale Transaction Processing applications, in practice there is a much more common

requirement. This is for large systems to support around 30 TPS, not with repetitive simple queries, but with a much more varied workload including complex queries. It is for this kind of system that a tool to help with optimisation could be very useful.

2.3 CCTA recommendations for performance prediction

A paper published by the CCTA (Central Computer and Telecommunications Agency, 1987) describes how performance prediction based on disk I/Os could be achieved using knowledge of the database system's architecture and query optimiser. It shows how performance prediction is used in the context of SSADM. A new SSADM form is proposed: Relational Database Access Calculations which the designer would use to do the calculations to optimise performance. The designer has to make his selections of likely solutions, and for each of them repeat the calculations to compare them for performance. SSADM is being promoted as a standard by the government, through the CCTA.

2.4 The need for Automation

It is stressed in various papers proposing automated systems (for example Finkelstein et al, 1988) that it is very easy to miss the best solutions, because there are far too many possibilities for them to be examined in a rigorous way. The very large number of alternative solutions possible for a physical design make an expert systems approach seem suitable, and this is what was adopted for this project. However a number of other approaches have also been studied.

2.5 An automatic index selector using broadly based costs

The relational database performance optimisation model described by Motzkin (1987 and 1985) compares overall costs for a system, when various alternative sets of indexes are applied to the files.

The cost function is very broadly based and takes into account disk space used, searches involved in queries, modifications, and reorganisations. Cost is given in time and in dollars.

Motzkin's system requires the database designer to enter a lot of information. This includes (a) system and operating environment constraints and costs, including total available disk space, time available, disk access times and average cost per access, (b) database information: the number of files, and information about each file, (c) workload information such as the number of searches and changes to each field of each file which the user will make per day, and (d) index information: the number of indexes which are available on each file, and the variations which the designer is thinking of.

2.6 Design-by-Example system to get pattern of queries, and select indexes

Design-by-Example or DBE (Bitton et al, 1985) is a system which borrows from a technique used in some query interfaces, Query-by-Example or QBE. The physical design it produces is based on example queries which it presents on screen. The designer modifies these to show what the user is likely to do.

A series of example tables is displayed, containing sample data, and example skeleton queries. These can be modified, and values inserted to define selections and joins, together with weightings that indicate the relative frequency of each query. DBE then generates candidate physical design schemes, and computes performance forecasts. For each design proposed, it presents a forecast for the response time for every example query. The designer decides when a desirable scheme is reached.

Besides the DBE query generator, Bitton's system is interesting for its index selector in which a search strategy is used to find the columns of a table which it is useful to index. For each table it selects one column as the primary key, and others to be secondaries, using rules based on the distribution of data and the pattern of queries.

The index selector employs a search strategy to decide on indexes for a given table, using the total number of disk accesses required for the pattern of queries as the cost for comparing alternatives. Initially a column is chosen as the primary key. Then a secondary index is assumed to exist for all other columns. Cost calculations are done to find the effect of dropping each of the secondaries from the set. This may produce a list of columns which, if indexed, result in a cost increase (see Section 3). If this is so the most significant of them is dropped, resulting in a set of secondaries reduced by one. The cycle is then repeated, until it is found that none of the secondaries can be dropped without increasing the cost. The result is a recommended set of secondaries.

2.7 Generating query sets automatically from the logical design

Ryan and Carlis (1984) present a different approach for generating query patterns, by analysing the entities and relationships in the logical design. They assume that an entity in an LDS (logical data structure) has a semantic importance which depends on its relationships with other entities. They derive a set of representative queries by considering paths between entities, one of the criteria to identify central entities being the count of the number of arcs in the diagram which indicate relationships. They claim that this is accurate enough for a first-cut design and saves a lot of time compared with a manual method of doing it.

2.8 Use of expert systems

Two papers from France (Bouzeghoub et al, 1985a and 1985b) describe an expert system, written in Prolog, which generates the logical design for a

relational database system. The authors say that their system is envisaged as being part of a more complete system to cover business analysis and physical design as well. The system described in these papers offers help with designs which have incomplete specifications. For each design step it offers general or specific principles of reasoning, and explanations. It allows back-tracking to allow changes to the specification, and new information.

2.9 An IBM experimental physical design tool

An IBM experimental physical design tool for relational databases, called DBDSGN, is described in Finkelstein et al, (1988). Given a particular workload (SQL statements and their execution frequencies) DBDSGN suggests a physical configuration for efficient performance. Each configuration consists of a set of indexes and an ordering for each table. DBDSGN reduces its search space by compiling a list of "plausible" columns for each SQL statement. One reason which defines a column to be plausible is that (a) there is a predicate on it and (b) the system can use an index to process that predicate. This happens when the predicate is ANDed to the rest of the WHERE clause, and it is usable as a search argument to retrieve rows through an index scan. If a table is not mentioned at all in a statement, or if columns are only referenced in the SELECT lists and never in the WHERE clauses, then columns are implausible for that statement. Having compiled its list of plausible indexes, DBDSGN performs a process of index elimination, and generates solutions.

An important principle of DBDSGN is to make use of the *same optimiser* as is used for run-time queries, i.e. the database management system optimiser. DBDSGN runs the optimiser to determine which columns might be worth indexing and to estimate the costs of execution statements in different configurations. This provides a very reliable basis on which to choose solutions because the optimiser is able to supply the estimates of cost it uses itself in order to decide on an access path. The optimiser examines the set of access paths that exist and computes the best expected cost for a statement by evaluating different join orders, join methods, and access choices.

3 Physical design for an Ingres relational database

Physical design for a database follows logical design, and concerns the physical structure of the files and indexes holding the data. Good performance from the database depends on good design, and physical design in particular. With Ingres a high degree of tuning can be achieved because of the options available, but experience and skill are required to make the best selections. In fact a rigorous search for the best design among all possible ones cannot be done in practice because of the number of possibilities. Part of the project was to formulate a set of rules for tuning, emulating what would be done by an expert.

The problem faced by the database designer is illustrated by considering the following list of things which have to be taken into account:

- the size of a database table: how many columns and how many rows?
- how often data is to be added, how fast the table is growing,
- whether disk space is at a premium,
- the general pattern of queries, including relative frequencies of various types, for example to search the entire table, to make fast access to a small number of rows, make joins with other tables, look for patterns and ranges of values,
- (for VME Ingres) the availability of Contents Addressable Filestore, CAFS.

An Ingres relation or table is physically stored in a file, held in pages which are 2048 characters on Unix, and 4096 on VME. A page is the smallest unit of retrieval from the disk.

Files can be set up with keys, in which case queries may make use of a key in finding tuples (rows) which satisfy a predicate of a query (i.e. a condition following a 'where').

When a query is entered, the Ingres Optimiser works out an optimum access path, from knowledge it has of the physical structure for the table, including its indexes, and its size. On VME with CAFS the optimiser will select CAFS if it offers the fastest retrieval, which will not always be the case, especially with a very big file when use of an index can be faster.

The most significant factor affecting the response time for a particular query is the number of disk accesses made. Other factors such as processor speed, or the relative speed of input/output to disk for one processor compared with another, are of second or lower order significance. The response time will be improved if for example Ingres is able to retrieve several records in one access rather than retrieving them one at a time.

Hence to minimise the number of disk accesses for frequently used queries is a major objective of optimisation.

The investigation carried out considered an Ingres system which does not make use of CAFS, and the prototype does not have rules which take into account its availability. However it is envisaged that the design used could be modified by adding appropriate extra rules to calculate the disk access cost of a query, but the rules for a search for alternatives in design might stay unchanged.

There are four storage organisations allowed: heap, indexed-sequential (isam), btree, and hash-random (see Appendix). All except heap are keyed structures which require a primary key. All structures can be "compressed", that is stored without trailing spaces, thus making eight structures possible. All can have extra keys which are called secondaries. These are held in separate files for which there are also alternative storage structures.

The availability of these alternatives allows considerable scope for 'tuning', which can dramatically improve performance.

The Ingres database can be set up by making all the tables heap, when no choice of key at all is required. Alternatively a primary key can be chosen for each table, with one keyed storage structure such as btree for all the tables. This may be quite adequate. However, in order to obtain the best performance possible, the physical design has to be optimised by selecting separately the best storage structure for each table, and then perhaps adding secondary keys.

For one table there is a very large number of feasible alternatives even when there is only a small number of columns. This can be appreciated by considering the variations possible. There are eight storage structures to choose from. Six require a column (or a combination of columns) to be chosen as the primary key. On all structures secondary indexes may be added to help with particular queries. There are four file structures to choose from for the extra indexes. The optimum choice depends on the pattern of use, which initially has to be anticipated, and then monitored while the system is running, to help with later tuning.

IPDA makes quantitative comparisons which could not be made accurately by an on-the-spot designer. Although his experience might tell him which solutions offered most hope, there is research (Finkelstein et al, 1988) to show that the optimum solution may easily be missed, as the addition of indexes carries costs which have to be balanced against the gains they are aimed to produce. Update queries can be slowed down when extra indexes are added. Balancing costs against benefits is not simple.

The number of alternatives for all the tables of a large database is so large that they could not all be considered in a realistic timescale. Carlis et al (1983) give a figure of 13 centuries with a Cyber 74 computer to look at all the possibilities for one (US Navy) database.

Other physical factors such as the placing of database files on the disks to minimise disk head movement, and the use of cached memory, also affect performance, and would be considered when setting up an Ingres system. Work on this would be done after the physical design and would be configuration-dependent.

4 Evolution of prototype system: 'Ingres Physical Design Adviser' (IPDA)

IPDA was evolved with the help of expert systems methodology and went through two prototypes. Given that the physical design process is largely a matter of an expert applying experience and "rules of thumb", and yet the problem seemed to demand some precise numeric calculations, a lot of heartsearching went into deciding whether an expert systems approach was indeed the most appropriate. The second prototype which eventually emerged has a sub-system to evaluate a cost which is measured in disk accesses, and a sub-system which searches alternative physical designs, and compares them and makes selections on the basis of this cost. As the project

evolved, this algorithmic approach and search sequence were seen to be essential. This implementation concentrated on this approach rather than on making use of an existing shell which would have had explanation facilities and a facility to enter new rules in an easy-to-understand form, rather than in Prolog. The prototype uses information about the database which is captured in the ICL Data Dictionary System, and a set of data was used to simulate an interface.

5 Description of IPDA, the prototype adviser

5.1 Outline

The prototype, IPDA, advises on the setting up of the file for one Ingres table. It reads some information which has been entered about entities and attributes (in a form which could be captured by a Data Dictionary), and it asks questions interactively from the designer about the pattern of queries to be made using the table.

IPDA uses the information it has available to make a search for a few of the alternative configurations from among the very large number of possible ones in order to find a solution to recommend. This is carried out in five steps, each of which settles on one variation and feeds its result to the next. The first step makes a list of the columns which IPDA considers worthwhile "candidate" keys, the second chooses one of them as primary key, the third chooses a basic (keyed) storage structure, the fourth eliminates any remaining keys from the list which appear to be more trouble than they are worth, and the fifth considers all those still left as secondaries, and finds the best storage structure for each of them. The end result is a set of indexes and structures for the files associated with the table, and it constitutes the IPDA's recommendation. (Compound indexes are not considered, nor is a heap structure with secondary indexes – this would be work for a further development.)

This "heuristic" search follows a sequence which seems likely to produce a sensible solution: perhaps an expert in the field might go about his design in a similar way. There is no attempt to examine all the possible alternatives, which would not be practical. At each stage only a small number of alternatives have to be considered.

IPDA employs an algorithm for calculating the "cost" of a set of enquiries on one database table, in terms of disk accesses, given a particular physical configuration. It uses this calculation several times in each of the search steps, in order to compare the costs for alternative configurations given the same query load.

IPDA does not deal with queries involving joins between tables, which would be an essential feature for a system like this (a method for doing this was discussed but there was not time to implement it). The use of compound keys might be decided on by a designer, but this was thought to be too

difficult to try to automate. Queries which insert records as opposed to updating them were also not estimated for in the algorithm for the cost function.

5.2 Functional description

IPDA produces a file organisation for *one table* based on the minimum disk-access cost found from a set of solutions considered. The details it gives are:

- if at least one key to be used, then one column as Primary Key,
- the Primary File Organisation: heap, isam, hash, or btree,
- a set of columns as Secondary Keys together with the File Organisations of each one (the same alternatives).

The search sub-system works through 5 stages of successive refinement of physical design evaluating alternatives using a cost function for disk accesses for the load.

The pattern of use for one table is obtained by asking for information about queries on it (Section 5.5 considers the user interface further). The information required is:

- the frequency in relative terms (say per second) for each of the set of queries,
- for *each query*:
 - whether it is retrieval only, or involves update,
 - whether there is only one predicate (where-clause), or if not how the predicates are combined, by conjoin (and) or disjoin (or),
 - for each of the *predicates* (Condition-clauses following ‘where’):
 - the name of the column involved,
 - whether it is updated (if an update query),
 - the number of records which will match the condition (an estimate),
 - the type of match, i.e. exact (an equals condition), a pattern, a range of values (e.g. between two values), or some other such as “less than”.

Note that only a limited set of conditions is handled. Joins between tables are excluded, as are aggregation, and complex boolean combinations of predicates.

It might be argued that it is artificial to require the designer to supply details of all predicates given. However manual methods such as SSADM currently require this. The only viable alternative is for the system to collect the statistics itself during a period of time when the database is being used by the user in his normal work. However these statistics would measure the queries actually being executed, not the queries users would like to execute if the performance were adequate.

5.3 The search strategy

The object of having a search strategy is to narrow the search space by making selections in stages, in which a basic choice is made in preparation for the next stage, from a few likely options.

The sequence is based on an expert's idea of what would work in practice (and it is interesting that a similar sequence is found in (Bitton, 1985) – see Section 2).

Apart from Step 4 the algorithm is linear, and Step 4 has a maximum number of cost calculations of $1/2 \times (\text{Number of Columns})$ squared. This is thought to be acceptable.

The algorithm rules out the alternative of heap with secondaries and no primary. This would be relevant if the system were to model insert queries as well as updates, which would be an added complication which was not considered.

The steps are as follows:

Step 1: Get the First Index List

- (a) Make the Organisation for the file heap, without any secondary indexes. Calculate the cost.
- (b) Calculate the costs for each column mentioned in at least one predicate, when it is a secondary btree index, with no other indexes. The file organisation is still heap. In each case where there is a reduction compared with the unindexed cost, then put this column on a list of indexes.

The result is to produce a list of those columns which, if used as btree secondary indexes on a heap table, would give a cost reduction in the absence of any other indexes.

The remaining steps are only carried out if there is at least one column on this list, otherwise the organisation must be heap without secondary indexes.

Step 2: From this list choose one column as Primary key, and make the others Secondaries

- (a) Calculate a set of costs, for each alternative of one column on the list being the primary key, and the rest on the list secondaries. (Assume btree organisation at this stage.) Select the lowest cost alternative. This gives one of the columns as primary key.
- (b) Select the lowest cost alternative. This gives one of the columns as primary key.

Step 3: Find the best Primary Organisation

- (a) Make all the Secondaries btree.
- (b) Calculate the costs for each alternative Primary Organisation for the file, i.e. hash, btree, isam. (Having selected a primary key, a keyed structure is implied and heap is not considered as an alternative).
- (c) Select the alternative with the lowest cost.

Step 4: Get the optimum list of secondaries

- (a) Keeping other factors constant, try dropping all the Secondaries one by one from the list output from Step 2 and calculate the cost, making a list of costs.

- (b) Compare the costs obtained with the initial cost (output from Step 3), and if there is an improvement (or no change), drop the one which gives the best improvement.
- (c) Then repeat this cycle but compare each cost with the cost obtained in the previous cycle. Iterate round, dropping columns from the Secondary list, until no further improvement is obtained (or the result is that there are no secondaries).

Step 5: Find the best organisation for each secondary index

- (a) Starting with all the Secondary indexes as btree, set them one by one to the organisation which gives the lowest overall cost (from the list of organisations of hash, btree, and isam).
- (b) Once an organisation has been chosen for an index on a column, leave it set when considering the next indexed column.

Some comments

After one demonstration there was a suggestion made that Steps 2 and 3 should be combined, so that all organisations were considered for each primary key. This was considered but it was thought – pragmatically – that selecting the primary key on the assumption that all keys were btree was likely to yield a good answer in almost all cases.

The system does not allow the user to intervene: that would make sense if the system came up with two alternatives which were close compared with others which it dropped. There might be an advantage in a system which allowed the user to impose his or her own choices, within a range offered by the system. It would also be a useful feature if the system could pursue two or more alternatives through successive steps.

5.4 Cost Function

This is evaluated by firing rules from the Knowledge Base of IPDA for calculating the number of disk accesses which will occur for the predicted usage pattern for one table. The inputs are as follows:

- the names of all the columns of the table,
- the total size of one row of the table, in characters,
- the set of queries on the table, and their frequencies,
- the basic file organisation: whether heap, hash, isam, or btree,
- if a keyed structure the column which is primary key,
- other column or columns on which there are secondary indexes,
- the file organisations for each of the secondary keys,
- the number of rows in the table.

The total cost in disk accesses is calculated.

The cost algorithm for a query uses two simple models:

- (a) to decide the access path the optimiser will choose,
- (b) to evaluate the cost of this access path.

The assumption is made that at run-time Ingres will use the indexes where it can make a faster retrieval. As we have discussed, the selection of access path is made by the optimiser, which will also select CAFS rather than indexed retrieval if it is present on the system, and if this gives faster retrieval. The cost function used in IPDA does not include consideration of CAFS. The algorithm would have to be refined for a CAFS system.

5.5 User Interface

IPDA uses a simple question and answer dialogue. It was not an aim of the project to produce an interface beyond one which was sufficient to demonstrate the prototype.

IPDA displays a list of tables which are in its knowledge base, and invites the designer to choose one for which he wishes to set up files and indexes. With knowledge of the table structure (from the logical design) IPDA then asks a series of questions about the pattern of queries on it, and the number of rows. (Some volumetric information could potentially have been retrieved from the data dictionary.)

IPDA prompts for details from the user, to get the frequency in relative terms (say per second) of each query, and a description.

Ideally IPDA would work from the SQL query text, but to avoid writing a query parser in the prototype, the user was asked for significant attributes of the query directly.

IPDA explains its reasoning, and displays its solutions as follows:

Step 1

- Display Cost for Heap Organisation, Unindexed,
- Display Cost Reduction for each column if used alone as an index,
- Display as candidate indexes the columns which give a reduction.

Step 2

- Display costs for each candidate index chosen as primary key, the others being secondaries (file organisations btree),
- Display as primary the one giving lowest cost, the others secondaries.

Step 3

- Display costs for each keyed organisation (isam, btree, hash),
- Display primary organisation chosen,
- Display cost.

Step 4

- Display costs for each secondary being dropped,
- State whether any cost reduction obtained,
- If so display the secondary being dropped (the one giving the lowest cost), otherwise display the secondaries selected as the full list, and end,
- Repeat.

Step 5

For each secondary key:

Display cost for each file organisation: isam, btree, hash (with the others either btree, or what has previously been selected in Step 5),

Display the organisation chosen for this key.

(Remember that the cost is still the cost for the pattern of queries, and that everything else about the table files has been fixed.)

6 Some conclusions

This prototype showed the principles of a database design adviser, in which an expert system approach was used to develop a knowledge base. Some of the research papers suggest possible ways for further development.

If the system were developed to have a complete rule-set, an expert system might then not be the best approach. There is a big advantage in using Ingres's own optimiser, called from the system (Finkelstein, 1988, see comments below). An IPDA software system which ran in the same environment as the Ingres database system could be used for the physical design, using the Ingres optimiser to compare costs of the alternatives it evaluated during its heuristic search.

Finkelstein et al stress that a global solution cannot be obtained for each table independently. Any index decision made for one table may affect the best index choices for another. The rule-set has therefore to take into account joins, and also, to be complete, insert queries, compression as a further option of file storage form, and perhaps the use of heap storage form plus secondary indexes. (For VME the cost function needs to be refined to take account of CAFS, but the Ingres optimiser will be able to do this.)

In comparison with IPDA, Motzkin's system takes into account a much wider range of factors. These are real considerations for a database administrator and an ideal system should also include them.

IPDA requires the designer to input information on the set of most-used queries. Bitton's DBE shows a useful interface technique for this. Ingres may at some future date collect statistics of its own which would make it unnecessary for the designer to enter estimates of usage. However this would not be wholly satisfactory because there would be no information on the queries which the user would like to enter but does not because the current design does not support them easily.

As the search method is untested in IPDA it seems promising that it is similar to Bitton's, though it was arrived at independently.

Finkelstein et al write in their conclusion "A tool should not use an independent model of the behavior of the underlying system, even if that model is more accurate than the system's internal model. Instead the

database system should export a description of its behavior ... Improvements in formulas and statistics should be incorporated into the optimizer, not into tools." Considering IPDA and Ingres, the Ingres optimizer is potentially able to supply the cost of its chosen access paths for a given configuration and query. If the optimizer were available to IPDA as a "cost module", then it could be used for this purpose. IPDA could supply it with one solution at a time, and then run the queries in the query load. This is potentially an excellent solution, and would do away with IPDA's own cost function sub-system, but keep the search sub-system. There would not be the possibility of IPDA selecting an option which (even if it were theoretically better) actually takes longer because Ingres does not choose the access paths predicted.

Finally there is potential in the ICL Data Dictionary for applying IPDA-type rules. The automatic generation of the Ingres database from the logical design would use volumetric and frequency information to define appropriate file storage forms and indexes.

Acknowledgements

I am grateful to Dr Michael Kay of ICL for ideas which I have used in my prototype system, and for assisting me with the project generally.

The project was sponsored by ICL as part of an MSc degree course in Information Systems Design at Kingston Polytechnic, and I am grateful to Tony Sherwood and others of Management Systems Centre, in Mainframes Systems Division, for allowing me the time to complete it. I also wish to thank the colleagues in ICL, and Dr Petros Gelepethis and other staff at Kingston Polytechnic, who helped me at various times.

Appendix

A note on Storage Structures

A storage structure is a file arrangement providing a way to access data in a table. Keyed storage structures provide a way to get to a particular row or set of rows within a table more quickly than if the file were unstructured.

Heap is the default storage structure used when a table is created by INGRES. It means that there is no key at all to the table, it is just a heap of data. A record is added at the end of the heap.

Hash storage structure is the fastest access method for exact match queries with no pattern matching. The index is a field chosen as the key. A hashing algorithm is used on the key to decide where to find the record on the disk. When the structure of a table is modified to hash INGRES distributes the data evenly over pages of store. The table is built by placing each record on the page where its key hashes to.

Isam (Indexed-sequential storage access method) storage structure also requires a key to be specified. It is more versatile than hash and supports pattern-matching, range scans, and partial key specifications as well as exact match retrievals. Isam tables use a static index that points to a static number of pages. The table is sorted by keys at the time it is set up, and an index is built up. This points either to other index pages, or to the data pages where rows with that key range are stored. Hence there can be additional disk accesses to scan indexes, unlike hash.

INGRES Btree (Balanced Multi-way tree) is the most versatile structure. It supports keyed access, range searching, and pattern matching. The index is dynamic rather than static, allowing the index to grow as the table grows, eliminating overflow problems which occur with isam or has (additional pages of index which have to be accessed indirectly, involving further disk retrievals). Index pages point to leaf pages, and these point to the records actually holding the data, which are in data pages. Because it is kept in balance all retrievals go through the same number of index levels. The major advantage compared with isam is that it eliminates overflow as the table grows (except where there are a large number of duplicate keys, i.e. having the same value). When an index page becomes full it is split into two, but a data record itself never has to be moved. As a table gets larger splitting occurs less frequently than when it is small, and it is usually at the leaf or lowest level.

References

- BITTON, D., MANNILA, H., RAIHA K.J, Design-by-Example: a Design Tool for Relational Databases, Report TR 85-692, Cornell Univ., Ithaca, NY, USA, July 1985.
- BOUZEGHOUB M., GARDARIN G, METAIS E, SECSI: An Expert System Approach for Database Design, First Conference on AI for Databases, Blanes, Gerona, Spain, Oct 85, (Barcelona, Spain: Consorci Inf. & Documentacio de Catalunya 1985), (1985a).
- BOUZEGHOUB M., GARDARIN G., METAIS E, Database Design Tools: An Expert System Approach. Proceedings of the Eleventh International Conference on Very Large Data Bases, 1985, Stockholm, 1985, (1985b).
- CARLIS J. V., MARCH S. T., and DICKSON G. W., Physical Database Design: a DSS Approach, in Information and Management 6, pp 211-244, 1983.
- CCTA, Performance Prediction for Relational Database, (Information Systems Engineering Report Number 10), CCTA, December 1987.
- FINKELSTEIN S., SCHKOLNICK M., and TIBERIO P., Physical Database Design for Relational Databases, ACM Transactions on Database Systems, Vol. 13, No. 1, March 1988, Pages 91-128.
- MOTZKIN D., An optimal physical database model, Math. Modelling (USA), 1987, Vol 8, pp 240-4.
- MOTZKIN D., Database performance optimisation, AFIPS Conference Proceedings 1985, NCC Conf, AFIPS Press 1985, pp 555-566.
- RYAN K. L., CARLIS J. V., Automatic generation of representative query sets, IEEE Comput. Soc. Press 1984, pp 262-70.
- Silver Bullet, Ingres Benchmark Report April 1988, RTI, and "Auditor Report Silver Bullet Benchmark", Tom Sawyer of Codd and Date Consulting Group, April 23 1988.

KANT — a Knowledge Analysis Tool¹

Graham E. Storrs

Logica Cambridge Ltd., Cambridge

Chris. P. Burton

ICL Knowledge Engineering, Manchester

Abstract

The project out of which this paper arises is a 5 year, £7 million collaboration between ICL, Logica Ltd, the universities of Lancaster, Liverpool and Surrey, Imperial College London and the Department of Social Security. The broad aim of the project is to help members of the general public in making claims under the Social Security provisions, and the staff of the Department in assessing the eligibility of these claims.

Achieving these aims involves assembling, manipulating and accessing a very large body of knowledge, much of which can change, some very quickly; and developing fast and efficient methods for accessing this knowledge. The present paper describes one of the main tools developed for building and maintaining this knowledge base.

Introduction

The Alvey DHSS Large Demonstrator Project is a £7 million, 5 year research project aimed at demonstrating the viability of intelligent decision support for large, legislation-based organisations. It is a collaborative project, partially funded by the UK Government, between Logica Cambridge, ICL, the Universities of Lancaster, Liverpool and Surrey, Imperial College, and the Department of Social Security (DSS).

The project is building three applications as demonstrator systems. These are:

- The Claimant Information System. This will provide advice and explanations to potential claimants of social security benefits as to their eligibility to claim and the consequences of claims they might make.

¹This work was carried out as part of the DHSS Large Demonstrator Project, supported by the Alvey Directorate of the UK Department of Trade and Industry and the UK Science and Engineering Research Council. The collaborators are ICL, Logica, Imperial College, and the universities of Lancaster, Liverpool and Surrey. The UK Department of Social Security is also actively participating in this work. The assistance of other project members is gratefully acknowledged. The views expressed here are those of the authors and may not necessarily be shared by other collaborators.

- The Local Office System. This is intended to provide decision support for DSS adjudication officers who are assessing claims for benefits in local offices. These people are legally empowered adjudicators and the decisions they must make are affected by large bodies of complex legislation and case law.
- The Policy System. This will provide support for DSS policy makers. The job of a policy maker is to formulate, explain and monitor the policy of the department, to implement it as social security legislation, and to modify existing legislation so as to reflect the current policy.

Knowledge Analysis in the DHSS Demonstrator Project

One of the prime objectives of the project is to investigate the use of very large knowledge bases. So it is with the local office (LO) system that we will be particularly concerned in this paper as it is only within this application that a very large knowledge base is being built.

While the project is faced with many problems of how to represent and organise very large bodies of knowledge within a machine for efficient inference, it rapidly became clear that the major difficulties with building large knowledge bases were in the analysis of the knowledge, the validation and verification of the knowledge base and the maintenance of the knowledge base. This despite the fact that the sources of knowledge for the systems we are building are, on the face of it, highly structured, definitional and rule-like.

A style of knowledge analysis has developed within the LO application. This is based on a restructuring of the knowledge in the various sources (Adjudication Officer's Guide (AOG) and the Acts and Regulations) into an essentially hierarchical breakdown of the concepts involved (e.g. the notion of "capital" in Income Support). Onto these, the "operation" of the regulations is superimposed as a set of linking rules and this "legal" perspective of the knowledge is systematically mapped to another structured knowledge base which represents the user's conceptions and the "task" knowledge. The task knowledge is unlike the legal knowledge in that part of it must be acquired through techniques such as interviews and simulations but most of it is also contained in the AOG. Figure 1 shows how the operational knowledge base is derived from these intermediate representations which are in turn based on the analyses and reformulations done by the analysts.

It is very difficult to say how large a "very large knowledge base" is as there are no adequate metrics. The target we have set ourselves is to have encoded about one third of the printed source material required for the adjudication of claims for income support (in fact, the project expects to have about half the source material encoded). This should encompass about two volumes of the eleven-volume AOG plus all the relevant acts and regulations as well as parts of some other sources (e.g. the Income Support Manual). Readers unfamiliar with these sources will not necessarily appreciate how large this body of knowledge is, nor how dense it is, nor how densely cross-referenced.

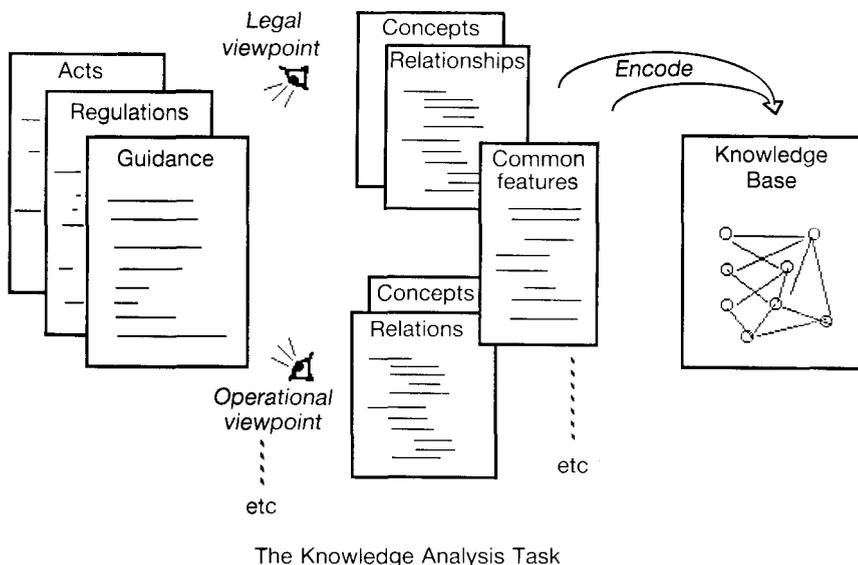


Fig. 1

It is the cross-referencing in particular that leads to extreme difficulties for the knowledge analyst. We have estimated that, in a typical month's work, each analyst will have referred to approximately 20 megabytes of text as a result of reference-following. This is a rather extreme instance of the "thumbs problem" and it is simply not possible for a person to keep track of this information unaided.

The Requirements for KANT

In order to support the knowledge analysis activity within the LO application, a hypertext-based tool, known as KANT, has been built. The requirements for KANT were:

- to support the project's knowledge analysis style directly – that is, to support the procedures for knowledge analysis with which the present analysts are familiar;
- to be easily usable by the present analysts – these are people with a legal or systems background without, necessarily, any computer skills;
- to be able to allow the user to work on more than one analysis at the same time;
- to have a sensible and, as far as possible, automatic route from paper source to KANT form;
- to be able to cope with the anticipated volume of material to be analysed – this involves having multiple sources (e.g. several acts plus the AOG) available at the same time;
- to handle this quantity of text without slowing down the analysis – we

took this to mean that browsing the sources should be able to be done at a pace similar to that of turning and scanning printed pages and that retrieving a reference by following a link should bring up the target text within a second;

- and, a requirement imposed by our users which caused great difficulties, that the sources should look, as nearly as possible, identical to their paper equivalents.

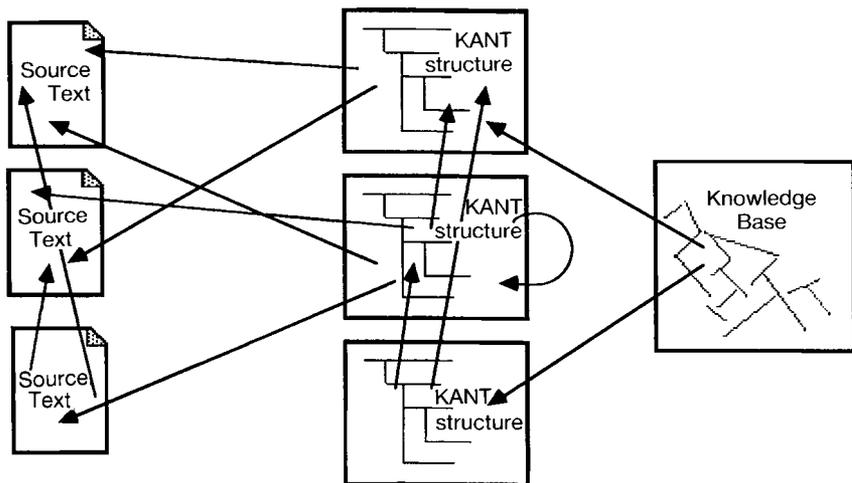
The Knowledge Analysis Tool (KANT)

Most of the design of KANT, and its implementation, has been done in the ICL part of the project team, as has most of the slog of turning paper sources into KANT sources. The implemented system is written in Interlisp on a Xerox 1186 Lisp workstation and makes extensive use of the project's screen management and text management software tools in order to gain the necessary performance.

KANT will allow the analyst to open up as many source documents as are needed. These each reside in a separate window and the windows may be moved, shaped, buried and revealed as suits the user. Sources in the context of the DHSS Large Demonstrator are such things as legislation, procedures guidelines and case law manuals. Sources may be browsed by scrolling at the rate of a line, paragraph, or page, or they may be searched for occurrences of strings. Similarly the user may open as many "structure" windows as necessary. These contain the results of the analysis. For each session, each node of each structure has "provenance" data attached. This gives the name of the analyst who is working on it, the reason for the analysis, the date and the time and a full record of all the significant changes to the node. Provenance records may be reviewed or printed as desired.

The structures created by the analyst are basically hierarchical and are usually displayed as indented blocks of text. A graphical "overview" is also available to assist rapid navigation over the structure. Structures may be folded and unfolded in the style of "ideas processors" or "outliners". Each node has its own internal structure. It has a title, then an arbitrary number of fields which are text strings of arbitrary length. The interpretation of the fields is determined by the conventions adopted by the analysis team, and typically they can be a body of free text, and a rules section (for adding rules in an "intermediate" rule language). Other software tools to do with Knowledge Base Building may make use of the conventions in order to extract the correct fields for input to a Rule Compiler, for example. A node may be moved to a new location within the same structure or copied to another location in the same or a different structure (in all cases bringing all its subordinate nodes and maintaining all its links appropriately). The analysts can obtain a hardcopy printout of a structure in any chosen degree of detail to allow offline study and discussion of the analysis.

Nodes in the structures and text in the sources may have links attached to



KANT-Links between items

Fig. 2

them. These links may go either to a piece of text in this or another source document, or to nodes in this or another structure, as indicated in Fig. 2. The links may be followed by buttoning on them, and doing so reveals the target text or node almost immediately (i.e. with a sub-second response time). The same link mechanisms are used to link entities in the operational knowledge base back to nodes in structures and text in sources. This can be used to call up justifications for user-guidance or diagnosis. The links may be named by the analyst and it is found that for each analyst, analyses proceed with relatively small numbers of link types (i.e. around 10). However, different analysts use different sets of link types and have commented that shared sets and standard common subsets might be useful additions to the system. The significance attached to the direction of a link is a choice for the analyst – a useful convention is that the origin object “depends upon” the destination object.

We have found that analysts like to work with several source documents open and several analysis structures open at the same time. Interestingly, some of these structures may be used as “scratch-pads” recording progress in the analysis and adding commentary and annotation to work in progress.

KANTification

The process of taking text and moving it into the KANT source format has, inevitably, become known as KANTification and it is a non-trivial problem. To give us the speed we need from the system, we handle text in blocks rather than character by character. These blocks must be pre-defined for the system.

They are, typically, about the size of a paragraph of text but may range in size from a full page to a single character. They not only include text from the body of sources but also marginal and footnotes, cross references and table entries.

The materials from which we cull our knowledge are occasionally found on magnetic tape in a standard mark-up language such as SGML. Such languages have sufficient information in them for us to specify the rules of KANTification and apply them to get a KANTified text. Some of the material is available on floppy disc (in more or less obscure formats). The process in this case usually requires some manual intervention to ensure that the word-processor conventions are correctly interpreted as properly formatted KANT text blocks (in particular the handling of footnotes and marginal notes can be problematic). However, a great deal of the sources we must deal with exist only in paper form. Using optical character reading followed by manual formatting has proved successful but tedious and expensive. A more cost-effective technique has been to have a word-processing bureau retype the documents and typographically mark them up as they enter them according to our specifications.

User experiences

Several of our analysts in the LO application (and, increasingly, in Policy and Claimant Information) have now used KANT in their work. The feedback from them has been largely positive. The major objective of using such a hypertext system to solve the “thumbs problem” has been met to judge by their reports. Problems with the system are largely to do with surface HCI features rather than the user’s conceptual model or the system’s support for the task.

One interesting observation is that users find they occasionally have difficulty in navigating around large analysis structures that they have created. The feeling seems to be that the physical appearance of the indented text blocks they create is more homogeneous than normal text so the usual cues to location are not present – even though the structures are strictly hierarchical. We do not yet have a complete solution to this problem – some users have solved it by adding ancillary structures linked to their analyses which act as guides or indexes. We do provide a graphical overview of the structure on which is indicated the user’s current location and which also allows the user to move rapidly to another part of the structure, but even this is of limited use with very large structures. The problem is exacerbated by the restricted size (1200 × 900 pixels) of what are by normal standards large high-resolution screens. A screen with four times the area would also be useful to enable several source windows to be browsable simultaneously.

The ability to create links and annotations within the source documents themselves is a facility much discussed in the hypertext literature but was not one which emerged as a requirement during the initial design of KANT. It

was called for after users had gained some experience with hypertext, and when the needs for control of the source amendment process became apparent.

Another interesting finding, already referred to, is that the analysts tend to create and use a relatively small number of link types. Some of these are for the object-level analysis of the source and others are for meta-level comment and structuring of the analysis process itself. Analysts seem to believe that a pre-defined set of links could be a useful addition to the system even though they tend to invent quite different sets for themselves. Such a set of links would, in effect, provide a basic epistemology of the domain and it is interesting to speculate how these sets of links might differ from domain to domain, how they might be employed in a distributed group analysis process in order to provide consistency in the analysis, and how their semantics might be defined for the system in order to provide a deeper understanding of the resulting structures and greater machine assistance with the analysis process.

Legislation is constantly changing. In particular the so-called secondary legislation (regulations) may be changing very rapidly. There is therefore a need to be able to update sources as they change, maintaining all the links that previously existed, adding new links (doing new analysis) where appropriate, and ensuring that changes in the source are reflected in the analysis and therefore the knowledge-base. Given the organisation in KANT of a source document into a "pool" of KANTified paragraphs, the document the user sees ("virtual documents") have an actual internal existence as ordered lists of references to the paragraphs in the pool. This provides the basis for the flexibility needed in the structure of a source document to support the maintenance of extensive and constantly changing material. Changes to a document are usually notified as a (sometimes very long) list of paragraphs which should be added to it or which replace existing paragraphs in it. To create a new version of a document, these new paragraphs must be added to the pool and a new virtual document created which is like the old one except that the new paragraphs replace or extend the old ones. Any hypertext links which existed for paragraphs in the old document which are no longer present in the new one must then either be appropriately connected to the replacing paragraph or deleted. KANT provides a "jeopardy" mechanism to assist the analyst in this task. When the destination object of a link changes due to editing, deletion, etc, the object at the dependent end of the link is automatically marked as jeopardised, with a prominent warning triangle superimposed on the image of the object. Thus the relevance of a change can be easily traced through to the analyses and any consequent changes can be propagated all the way through to the knowledge-base. What is more, because old paragraphs are not deleted from the pool and retain their original links, the old virtual document can be retained and thus a complete system of version control is possible.

The choice of the size of the node is an interesting parameter to consider.

KANT takes the position that each node is a “paragraph”. This designation is more to support the user’s conceptual model than it is accurate. A paragraph may indeed be a paragraph (in a source text for instance) but it may also be a footnote, a marginal note, or whatever. It may even be a single character if that is appropriate. However, in the analysis structures, the “paragraphs” are always structured, for example as title, text and rule, each of which has a different meaning to the system. As mentioned earlier, facilities exist for the user to define his or her own node structures as well as there being several other “library” structures available.

The choice of the paragraph as the “grain size” for the node has some consequences in use. For instance, some users have expressed the need to link to and from individual phrases within paragraphs and individual components of rules. The fact that this is not allowed sometimes leads them to create unnatural structures, breaking up their analysis in awkward ways so that the linkable chunks are each in a separate node.

Other systems tackle the grain size problem in different ways. The card-based systems, for example, generally use a whole card as the unit for linking to, while they allow single words or smaller units to be used to link from. Such choices seem to lead to different perceptions of the systems being used. Certainly there is something of the feel of using menus – albeit menus embedded in text – with the card-based systems, whereas text-oriented systems such as KANT retain much more of the feeling of reading linear text. In fact, rather as there is a continuum between hypertext and semantic networks, there is one between hypertext and menu systems. In this case, the dimensions which vary are: the size of the displayable unit (i.e. what constitutes the display of a linked-to node), the explicitness of the marking of links in the text, and the amount of embedding of the linked text in non-linked text.

Concluding remarks

Some of the most exciting and, indeed, appropriate applications for hypertext systems appear to be in areas to do with creativity, exploration and design. Knowledge analysis is clearly such an area and the hypertext system we have built to support this activity within the DHSS Demonstrator Project is proving to be an effective and useful tool, apparently well-liked by the project’s analysts and well targeted to their needs.

The use of KANT takes us only so far down the road towards a finished knowledge base. That is, it takes us as far as an intermediate representation of the rules and structures we wish to incorporate. Another tool, the Knowledge-Base Builder (KBB) is then used for knowledge encoders to take the KANT intermediate representation and turn it into rules and objects in the target knowledge representation language. Explorations are in progress to discover the extent to which the two tools can be merged into a single environment for analysing knowledge and building knowledge bases. One

great advantage of such a system would be the fact that individual knowledge base entities could then be traced back all the way to the original sources with a clear “audit trail”, provenance information and records of intermediate development decisions. For applications based on legal sources, which are constantly being revised, this traceability would be of enormous benefit to those tasked with the maintenance of the knowledge base.

Another interesting aspect to the development of a hypertext tool for knowledge analysis is that there appears to be a smooth progression between hypertext and some knowledge representation schemes which is potentially exploitable.

Hypertext, in its simplest form, is a set of nodes connected together by undifferentiated links. Each node is an unstructured piece of text or graphics (or both) and each link is a uni-directional association between two nodes. However, both nodes and links could have a lot more structure than this and their structure could have a lot more meaning for the system. For instance, the nodes need not be free text but could be structured so that there were a number of named text, graphics or even numeric fields. The nodes could also be of different types. The links too need not be simple associations but could be special types of relationship such as “is defined by”, or “supersedes”. There could also be meta-level organisation so that link relationships could be described in terms of their directionality, or their transitivity (a link type such as “supersedes” would be unidirectional and transitive, for instance).

In fact, as nodes and links become more and more structured and more and more meaningful, the nature of the system changes progressively from being hypertext to being a kind of knowledge representation style known as a semantic network. The best-known semantic network representations are those supported by the popular artificial intelligence (AI) toolkits such as Art, KEE, and KnowledgeCraft. These representations are also known as frame-based because the node is a “frame”, that is, a set of labeled and, perhaps, typed slots for information – rather like a record in conventional programming languages. Some of these slots may hold a reference to another frame and thus become, effectively, relationships (i.e. links) between nodes.

Thus we have a continuum between hypertext and knowledge representation schemes which varies along the two dimensions of node and link structure. The existence of this continuum makes possible the opportunity to build intelligence into hypertext systems. The way to do this would be to add sufficient structure to both nodes and links so that the hypertext “document” may be interpreted as knowledge by a knowledge-based system.

Notes on the screen images

The following Figures 3–6 are images taken from the screen in a KANT session. The process of capturing has lost some of the detail – for example

typefaces appear much smoother on the screen, without 'jaggies', but the overall appearance is realistic.

Figure 3 shows an overview of most of the screen. The analyst has opened two sources, and is working on a single KANT structure. The KANT control area is near the top right of the screen. The need for more screen area is apparent.

Figure 4 shows a section of a source window. One paragraph has been highlighted as a result of the operator 'buttoning' on the end of a link in a structure. This and another paragraph have a small arrow icon attached to them showing that they are the destination ends of links from structure nodes.

Figure 5 shows a section of a structure window. Nodes with a '+' icon are 'unfolded', the children of the node are visible. Nodes with a '-' icon are folded, they contain descendants which are not currently visible. Buttoning on the icon switches to the other mode. Nodes with a '0' icon have no descendants. Arrow icons leaving the node denote the origin ends of links, and incoming arrows denote destination ends. Upper arrows refer to sources, lower arrows to structures. Buttoning on an arrow causes the object at the other end of the link to be scrolled into view and highlighted, possibly via a selection menu in the cases where one arrow represents several links.

Figure 6 shows the same structure fragment, in the case where the destination end of one of the outgoing links has been modified, by editing, for example. The node is jeopardised, and the analyst is thereby warned to inspect the situation and possibly take some action.

In Figures 3 and 4, Crown copyright data is used by permission of the Controller of Her Majesty's Stationery Office.

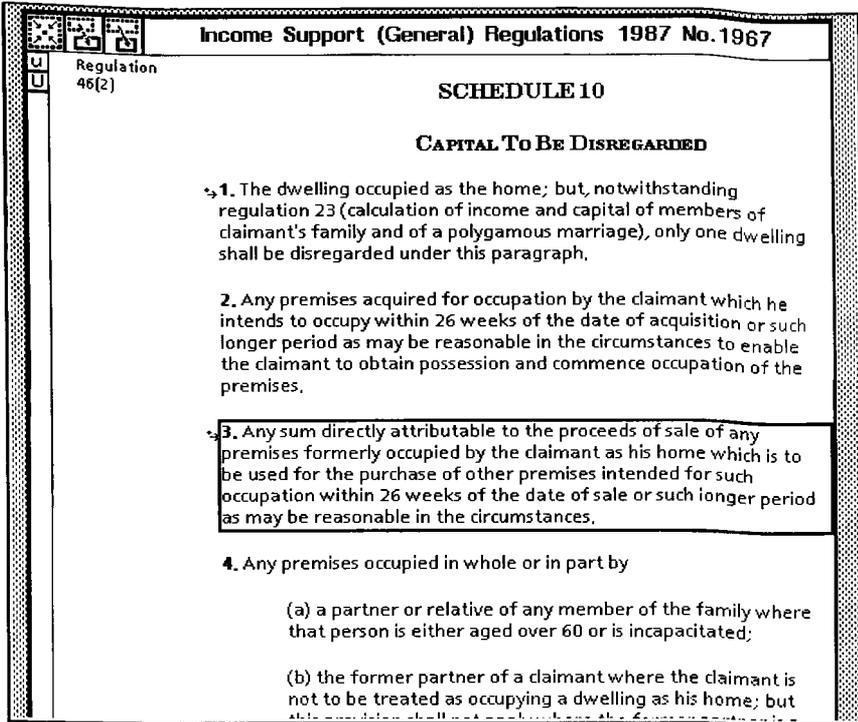


Fig. 4

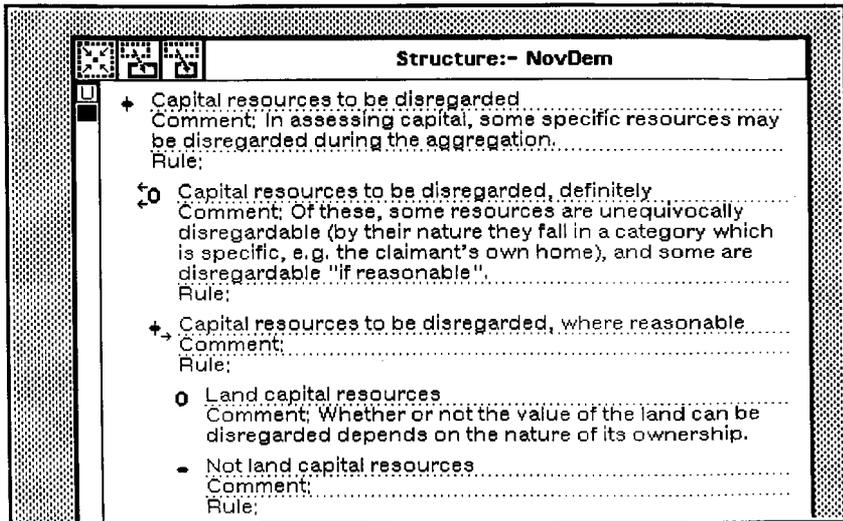


Fig. 5

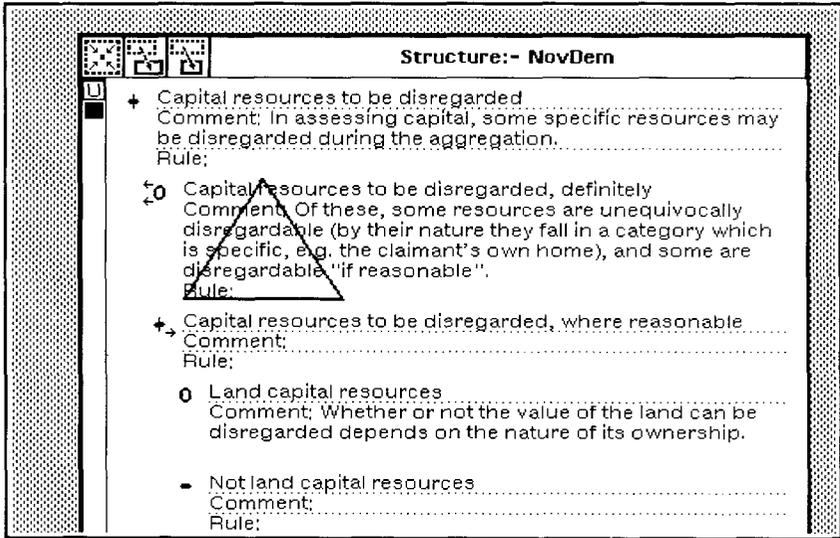


Fig. 6

Pure logic language

Edward Babb*

ICL Future Systems, Bracknell, Berks.

Abstract

ICL's aim was to design a logic programming language to simplify the creation of application packages and therefore lower the cost to the end user of solving his problems. Modern Database systems map the user's query to fast search software. Our strategy was to generalise this – and design a language whose main purpose was to help with the mapping of a users problem to conventional fast search software. The result, ICL's Pure Logic Language, performs this mapping in a general fashion. ICL's Language is briefly described and applied to some simple applications in electronics, mechanics and business. It is also succinctly compared with some other state of the art logic languages being developed elsewhere.

1 Introduction

This short paper describes briefly the philosophy and nature of a Pure Logic Language **PLL** being developed in ICL Bracknell. Some features of PLL, such as *metalevel reasoning* and *interval reduction* are only now being implemented. These and other features of the language, have therefore, not been fully described. In addition the examples chosen have been deliberately simple so that a range of applications can be illustrated rather than one application in depth. In fact, the language has been applied to two major applications – PAYE and Distribution Manager *ref 10*.

ICL's long term goal is to make computers more *concise* and *correct* to use by means of logic:

The *conciseness* of logic can be explained by a simple example. Suppose we define a relationship between three variables using the formula: $x = y * z$. Depending on how we use this equation it corresponds to the algorithms: $x := y * z$, $y := x/z$, $y = z \ \& \ y := \text{sqrt}(x)$. As you can see, the conciseness of logic arises because many algorithms can be described by one concise logical formula.

The improved *correctness* of logic, in building applications, derives from this conciseness. Since the descriptions are now more mathematical, we

*Supported by ICL and Alvey grants IKBS 084 & 092.

can employ mechanical theorem proving, to check the consistency of our rules against more general common sense rules – and hence try to stop the entrance of invalid data and rules.

1.1 Design strategy

The Pure Logic Language was designed according to the following basic ideas: *ref 5,6*

- Logic expressions are always simplified back to logic expressions. This means that some answer will always be returned by a query in the language.
- Unsolvable expressions should terminate. Important in large systems where infinite looping can be very expensive and difficult to distinguish from real problem solving *ref 3,4*.
- *unsolvable* and *FALSE* expressions must be distinguished¹.
- The expressiveness of full classical logic² is provided. Greater expressiveness improves the chances of the user expressing himself correctly.

2 Elementary operations

2.1 Arithmetic

We can write the following query in PLL:

1.84 * pounds = 14?

and get the answer **pounds = 7.6087**. In this paper the query is written before a question mark and the answer is written to the left of a *rewrite* symbol –R–>.

We can also write this query in equivalent forms and still get the same answer:

1.84 * pounds – 14 = 0?
–R –> pounds = 7.6087

14/18.4 = pounds?
–> R –> pounds = 7.6087

In each case, an attempt is made to find a value for the variable *pounds*. This flexibility in the language, means that the user is freer to choose his natural style, rather than the machine's. Furthermore, we can string statements together, connected by an *and*. For example: *1.84 dollars equals one pound,*

¹ PROLOG treats *unsolvable* as *FALSE* and therefore *not-unsolvable* becomes *TRUE!* which by classical logic is incorrect.

² PROLOG is restricted to Horn clauses.

and given *there are 14 dollars and the cost of an object minus thirteen is equal to fifty six times the unit dollar price*. This can be written:

$$\begin{aligned} 1.84 * \text{pounds} &= \text{dollars} \ \& \ \text{dollars} = 14 \ \& \\ \text{cost} - 13 &= 56 * \text{pounds} \ -R \ -> \\ \text{dollars} = 14 \ \& \ \text{pounds} &= 7.6087 \ \& \ \text{cost} = 439.08 \end{aligned}$$

Because PLL is based on logic, we can only assign to the variable *dollars* once.

2.2 List operations

Like LISP and PROLOG, the language allows operations on list structures. To split a list into its *head* and *tail* we use the equation $list = headoflist :: tailoflist$. Thus the following:

$$\begin{aligned} [rose \ 'daffodil \ 'daisy] &= Xh :: Xt? \\ -R \ -> \ Xh &= 'rose \ \& \ Xt = [daffodil \ 'daisy] \end{aligned}$$

splits the list $[rose \ 'daffodil \ 'daisy]$ into the head *rose* and tail $[daffodil \ 'daisy]$: alternatively, we can equate³ two list structures where some of the elements are variables:

$$\begin{aligned} [a \ 'daffodil \ 'daisy] &= (rose \ b \ 'daisy)? \\ -R \ -> \ a &= 'rose \ \& \ b = 'daffodil \end{aligned}$$

2.3 Range operations

The equation $x \text{ in } S$ allows us to generate or test membership of a list:

$$\begin{aligned} x \text{ in } [rose \ 'daffodil \ 'daisy]? \\ -R \ -> \ x &= 'rose \ \text{or} \ x = 'daffodil \ \text{or} \ x = 'daisy \end{aligned}$$

or range of numbers:

$$\begin{aligned} x \text{ in } [1 \dots 10]? \\ -R \ -> \ x &= 1 \ \text{or} \ x = 2 \ \text{or} \ x = 3 \ \text{or} \dots \ x = 10 \end{aligned}$$

2.4 Rewrite definitions

A typical predicate, defined by the user, has a predicate name, followed by a list of variables, and then an expression. A new predicate such as $convert(d \ p)$ is declared as follows:

$$\text{define } convert(d \ p) \ \text{tobe} \ 1.84 * p = d?$$

³The solution of sets of equations involving list structures is the equivalent of unification in PROLOG.

When *convert* is used in a query:

```
convert (pounds 14)?  
-R -> pounds = 7.6087
```

it is rewritten to $1.84 * \text{pounds} = 14$ with *pounds* & *14* being substituted for each occurrence of the parameters *p* & *d*. Finally, this expression is evaluated to give the answer shown.

2.5 Unsolvable problems

A key feature of the logic language is its behaviour on unsolvable problems as shown in these examples:

```
x = y + 20?  
-R -> x = y + 20  
  
not(x = 20)?  
-R -> not(x = 20)
```

Typically, $x = y + 20$ is unsolvable because there is an infinite set of values of *x* and *y* that satisfy this equation. In each case, unsolvable problems are just rewritten to themselves and not to FALSE⁴.

Alternatively, part of the problem can be solved:

```
y + 4 = 14 & f(x)  
-R -> y = 10 & f(x)
```

In this case $y + 4 = 14$ is reduced to $y = 10$, but $f(x)$ is left unchanged since f has no definition.

2.6 Other features

PLL has many other features *ref 9* such as *recursion*, *guards*, *and-rewrites* and further temporal operators. These are not described to keep this introductory paper on PLL simple.

3 Scientific applications

3.1 Introduction

The electrical circuit and mechanical system have been chosen to illustrate the operation of the language. The practical application of PLL for ICL would more likely be deductive databases and planning.

⁴As might happen in PROLOG.

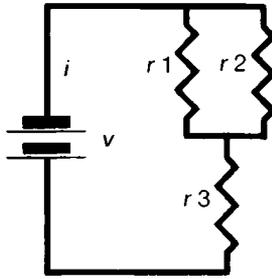


Fig. 1 Electrical network

3.2 Electrical circuit example

An electrical circuit, Fig. 1, consists of two resistances $r1$ and $r2$ in parallel, followed by a single resistance $r3$ in series with these. Across this circuit is a 4 volt lead-acid accumulator. Such a system is simply modelled by a set of equations. This can then be used in a flexible manner by a circuit designer, to design specialised solutions to other people's problems.

3.2.1 Parallel circuit: First, we define the resistance of any pair of parallel resistances $r1$ and $r2$:

define parallel_r(r r1 r2) tobe $1/r = 1/r1 + 1/r2?$

Using this definition we can compute the resistance of a 3 ohm and a 6 ohm resistance connected in parallel:

parallel_r(r 3 6)?

-R -> $1/r = 1/3 + 1/6$

-R -> $1/r = 0.5$

-R -> $1/0.5 = r$

-R -> $r = 2$ (ohms)

As this detailed⁵ reduction shows, the predicate *parallel_r* is rewritten to $1/r = 1/3 + 1/6$. A 3 and a 6 have been substituted for $r1$ and $r2$. The formula is then further reduced, using in built arithmetic rewrites, to its simplest form $r = 2$.

Unlike a traditional computer language, which computes in only one direction, the language can often operate reversibly. Thus, if we know the value of one of the two parallel resistances – namely 6 ohms and we know the total resistance is 2 ohms, then the language can still obtain the value of the other resistance:

⁵All this detail is not actually produced!

parallel_r(2 rx 6)?

$$-R \rightarrow 1/2 = 1/rx + 1/6$$

$$-R \rightarrow 1/rx = 1/6 - 1/2$$

$$-R \rightarrow 1/rx = 2/6$$

$$-R \rightarrow rx = 6/2$$

$$-R \rightarrow rx = 3 \text{ (ohms)}$$

Reduction takes place as before, except that a slightly different order is now taken.

3.2.2 Complete circuit: Next we define the resistance of any two series resistors:

define series_r(r r1 r2) tobe $r = r1 + r2$?

The total resistance is given by using these parallel and series predicates to form a single expression representing the complete circuit resistance:

define all_r(R r1 r2 r3) tobe
some(rx) parallel_r(rx r1 r2) & series_r(R rx r3)?

In this definition, we define rx from $r1$ and $r2$ using the parallel predicate. We then use the series predicate to combine rx with $r3$ to obtain the total resistance R . The quantifier *some*⁶ can be thought of as declaring rx as a local variable – invisible outside its scope. The complete circuit is then modelled by including ohms law and this *all_r* predicate:

define circuit(v i r1 r2 r3) tobe
some (R) $v = i * R$ & all_r(R r1 r2 r3)?

Because the total resistance is not required in the model it is also made local with a *some(R)* quantifier.

3.2.3 Using the electrical circuit: We can now use this 5 variable electrical model to calculate the current given a 4 volt accumulator and values of 3, 6 and 2 ohms for $r1$, $r2$ and $r3$ respectively:

circuit(4 i 3 6 2)?
-R -> some (R) $4 = i * R$ & all_r(R 3 6 2)
-R -> some (R) $4 = i * R$ & $R = 4$
-R -> $4 = i * 4$
-R -> $i = 1$

or to obtain the value of $r3$ assuming we know the current is 1 amp:

circuit(4 1 3 6 r3)? -R -> $r3 = 2$ (ohms)

⁶The same as \exists in classical logic.

However, suppose the two resistances are unknown but the voltage is 4 volts and the current is 1 amp:

$$\text{circuit}(4 \ 1 \ r_1 \ r_2 \ 2)? \rightarrow -R \rightarrow 0.5 = 1/r_1 + 1/r_2 \text{ (ohms)}$$

The query is now rewritten to an equation relating r_1 and r_2 , rather than a definite numerical solution. Now suppose, we know all the resistances values but not the voltage and current:

$$\begin{aligned} \text{circuit}(v \ 3 \ 6 \ 2)? \\ -R \rightarrow v = i * 4 \text{ (volts)} \end{aligned}$$

In this case, we get the relationship between the voltage and the current as our answer. If we now decide the voltage is 8 volts we can then use this equation $v = i * 4$ rather than the original circuit predicate to answer our question. Thus we have avoided the unnecessary recomputation needed when starting from the circuit predicate.

Each of these queries would need a separate algorithm in a conventional programming language. The conciseness of PLL means that we need only one definition.

3.3 Gear Conrod Piston

A mechanical system consists of two gear wheels connected to a conrod driving a piston in the manner shown in Fig. 2. This first gear is 1 cm radius, the second is R cms radius and this drives a conrod of L cms in length with a piston at its end.

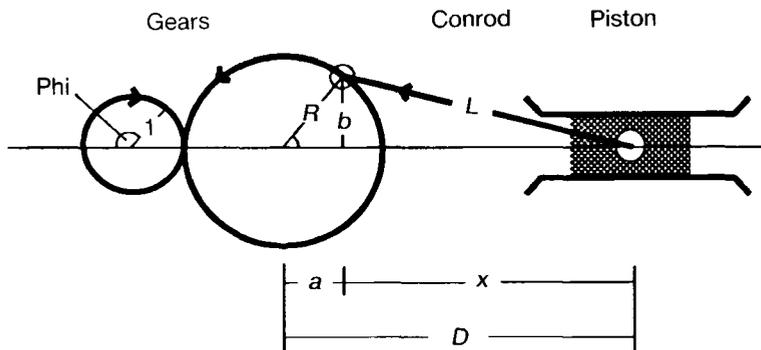


Fig. 2 Mechanical system

This is represented by an equation:

$$\begin{aligned} \text{define piston}(\Phi \ R \ L \ D) \text{ to be some}(a \ b \ x) \\ a = R * \cos(\Phi/R) \ \& \ b = R * \sin(\Phi/R) \end{aligned}$$

$$\begin{aligned} & L * L = x * x + b * b \ \& \ D = a + x \\ & L > 0 \ \& \ D > 0? \end{aligned}$$

Starting from *Phi* in Fig. 2, the value of *a* and *b* are obtained using $a = R * \cos(\textit{Phi} * R)$ and $b = R * \sin(\textit{Phi} * R)$. Pythagoras, $L * L = x * x + b * b$ gives *x* and *b* from the conrod length *L*. The position of the piston *D* is then the sum of *x* and *a*.

Using this mechanical model we can ask a variety of questions such as *what is the position of the piston?*

piston(20 2 4.5 D)? -R -> (D = 6.45619)

or *what is the length of the conrod?*

piston(20 2 L 6.0)? -R -> (L = 4.04532)

4 Commercial applications

4.1 Introduction

Often we are faced with the problem of handling uncertain information especially in the early stages of planning. One use for the PLL temporal database capability is in representing uncertain facts about people's holiday dates. We can then use this in deciding on dates of meetings or in persuading people to provide more definite information.

The next two applications involve commercial databases and show how complex data and queries can be represented in a Pure Logic Language. All these applications clearly need a friendly front end and PLL should be seen as the formalism for such a front end to map to.

4.2 Temporal logic

PLL has a temporal database capability based on Interval Quantification *IQ* ref 8,12. A simple use, for this temporal logic, is to model data about holidays in a small office. It can handle the usual uncertain information that normally precedes a definite entry in a holiday planner. Figure 3 shows that early in the year, we only know the following about Ed and Daves holidays:

Dave goes on holiday before Ed. Dave takes his holiday before 25th May. Ed must go on holiday before 1st June. Ed must return to work after 15th June and Dave must return after 17th June.

Figure 3 represents the slack in the position of these dates by the shaded areas. In PLL this time sequence is asserted by the following *add date1 before date2* commands:

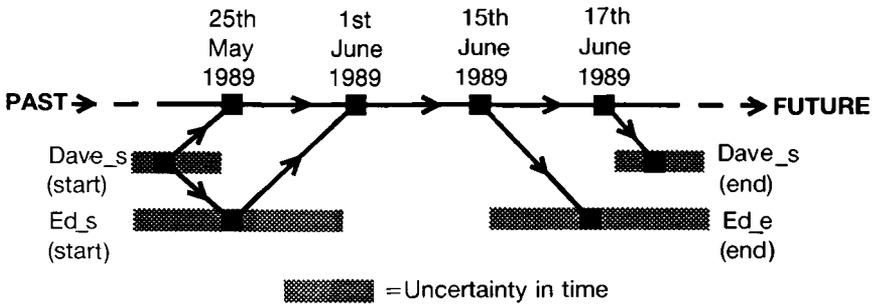


Fig. 3 Uncertain holiday dates

```

add 'Dave_s before 890525?
add 'Ed_s before 890601?
add 'Dave_s before 'Ed_s?
add 890615 before 'Ed_e?
add 890617 before 'Dave_s?

```

A predicate called *iand* which will perform high speed intersection of intervals is used to find if both Ed and Dave are on leave between the 8th June and 13th June:

```

I = iand([[ 'Dave_s 'Dave_e ][ 'Ed_s 'Ed_e ][ 890608 890613 ]])?
(I = [ 890608 890613 ])

```

or when are both Dave and Ed on leave before the 13th June?

```

I = iand([[ 'Dave_s 890613 ][ 'Ed_s 890613 ]])?
(I = [ "Ed_s" 890613 ])

```

or when are both Dave and Ed on leave?

```

I = iand([[ 'Dave_s 'Dave_e ][ 'Ed_s 'Ed_e ]])?
(I = iand([[ "Ed_s" "Ed_e" ][ "Dave_s" "Dave_e" ]]))

```

In this case we can see from the diagram above that there is no definite answer. The finish date of Dave's leave can be either before or after the finish date of Ed's leave. Rather than return a complicated conditional expression⁷, the original query is returned⁸. The user then can either alter his query or add more definite temporal information in the time sequence database.

4.3 Open and negative database information

Most databases use a closed world assumption where facts that are not true are assumed false. Pure Logic Language easily allows a more subtle

⁷Which can be done.

⁸Or a simplified form if some of the intersections can be unconditionally performed.

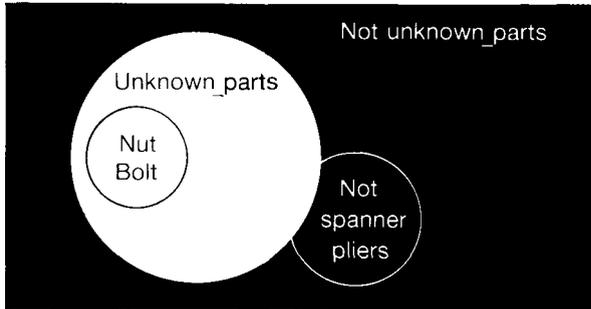


Fig. 4 Venn diagram of parts

representation to be used where this is appropriate. In the Venn diagram in Fig. 4, there are parts *nuts* and *bolts*, *unknown parts* and definite non parts such as *pliers* and *spanners*. This information is represented as follows:

```
define part_db(part) tobe
  (part in ['nut 'bolt] or
   unknownpart(part))
&
  ~ part in ['spanner 'pliers]?
```

If we now ask questions of this database, we get a more natural response. Thus, **Is nut a part?** is TRUE. **Is spanner a part?** is FALSE and most important **Is gear a part?** is unknown rather than FALSE.

```
part_db('nut)?      -R -> TRUE
part_db('spanner)? -R -> FALSE
part_db('gear)?    -R -> unknownpart("gear")
```

This corresponds to the answer we would get from *people* if we asked the above questions. In fact, even negated queries such as **Is a nut not a part?** give the correct answer:

```
~ part_db('nut)?    -R -> FALSE
~ part_db('spanner)? -R -> TRUE
~ part_db('gear)?   -R -> ~ unknownpart("gear")
```

4.4 Complex database query

Consider the following small database of suppliers and parts:

```
define part(p) tobe
  p in ['nut 'bolt]?

define sp(s p) tobe
  [s p] in [['s1 'nut]
            ['si 'bolt]
            ['s2 'nut]]?
```

Using this database it is possible to ask questions in the *Predicate Calculus*. Predicate Calculus was first used by the great philosopher and logician Bertrand Russell. Many database query languages are based on Predicate Calculus but use only *tuple variables* rather than *domain variables*. Even though PLL can use both, domain variables are usually the most natural: thus we can ask⁹ *Which supplier makes all parts?*:

```
all(p) (part(p) -> sp(s p))?-R -> (s = "s1")
```

Examination of the above database should show that indeed *s1* is the only supplier of all the parts. Alternatively we can ask¹⁰ *Which suppliers do not make all part?*

```
define supplier_all_part(s) tobe
  all(p) (part(p) -> sp(s p))?
  (some(p) sp(s p)) &~ supplier_all_parts(s)?-R > (s = "s2")
```

Again, it must be emphasised that PLL is not really intended for direct end user usage – rather it is meant as a language for interfacing to some friendly interface system with this formalism usually being hidden away!

5 Discussion

5.1 State of the art

It is beyond the scope of this short paper to provide a complete state of the art on logic programming.

John Lloyd in *ref 7* and in his unpublished lecture notes refers to many problems with traditional PROLOG. A typical problem is that $p(x), q(x)$ is FALSE whereas changing the order to $q(x), p(x)$ causes it to succeed. In fact such *unsafe* behaviour is no problem to PROLOG enthusiasts because they have a precise procedural model of the language which allows them to overcome such difficulties.

ECRC in Munich have augmented PROLOG with an optimiser *CHIP* meaning *Constraint Handling in PROLOG ref 12*. Many problems involve searching using many variables. If each variable ranges over many states, the search space quickly becomes impossibly large. CHIP avoids such coarse searching by essentially reasoning about the domains of values – in particular reasoning about the upper and lower bounds on values. CHIP also uses similar techniques to that used in database search algorithms where database style bit maps *ref 1,2* are used to store intermediate results.

⁹ $\forall p(part(p) \Rightarrow sp(sp))$ in predicate calculus.

¹⁰ $sp(sp) \wedge \neg \forall p(part(p) \Rightarrow sp(sp))$ in predicate calculus.

IBM in York Town Heights have a language called CLP (*R*) Jaffar et al *ref 10*. This is another enhanced version of PROLOG which includes an equation solver – they say *solving constraints in the domain of uninterpreted functors over real arithmetic terms!*

The Pure Logic Language described in this paper aims to solve some of John Lloyds objections to PROLOG and to include some of the constraint satisfaction ideas in the ECRC and IBM work. However, unlike these systems PLL is a rewrite language and has an altogether more general expressive power than PROLOG with its narrow reliance on Horn Clause syntax.

6 Conclusion

The interpreter is implemented in the computer language C. Expressions and other data structures are held using traditional pointer data structures. However, technical details of its implementation are well beyond the scope of this short paper. Performance of the language seems to compare with and sometimes exceed conventional interpreted PROLOG.

As mentioned earlier many features of the language have not been described in this introductory paper. Some crucial features are only now being implemented such as special predicates for metalevel control. Hopefully some more detailed paper will elaborate on these extra features.

ICL intend to exploit this research by making the Pure Logic Language the basis of a Visible Logic *Eureka* project called VisiLog. This aims to use the improved expressibility provided by logic as the language underpinning a spreadsheet/icon based system for helping people solve problems.

Acknowledgement

PLL is based on work performed mainly under ALVEY contracts *IKBS 084 Pure Logic Language* and *IKBS 092 Logic Database Demonstrator*. These contracts involved Imperial College, Edinburgh University, Bradford University and Turing Institute. The funding by Alvey and ICL and the contribution of Peter McBrien¹¹, David Cooper¹², Pete Slessenger and Joan Travis is gratefully acknowledged. In addition, we received background help from groups led by Professor B. Richards at Edinburgh University, Professor Dov Gabbay at Imperial College and Professor Imad Torsun at Bradford University.

¹¹ Most of the PLL implementation *ref 9*.

¹² Temporal logic *ref 8*.

References

- 1 BABB, E.: *Performing Relational Operations by means of Specialised Hardware*. ACM TODS. March 1979.
- 2 BABB, E.: *File Correlation Unit*. ICL Technical Journal. Nov. 1985.
- 3 BABB, E.: *Finite Computation Principle: An alternative method of adapting resolution for Logic Programming*. Proceedings of Logic Programming 83 – Portugal. June 1983.
- 4 BABB, E.: *The Logic language PrologM in database technology and intelligent knowledge based systems*. ICL Technical Journal. Nov. 1983.
- 5 BABB, E.: *Requirements for Large Knowledge Bases*. ACM 84 – The challenge of the 5th generation. September 84. San Francisco.
- 6 BABB, E.: *Mathematical Logic in the Large Practical World*. ICL Technical Journal. Nov. 1986.
- 7 LLOYD, J.W.: *Directions for metaprogramming*. Bristol University report.
- 8 COOPER, D.: *Temporal Operators in PLL*. SSC internal report.
- 9 McBRIEN, P.J.: *PLL User Version 0.32 Issue A*. SSC research report.
- 10 SLESSENGER, P.H.: *Trial Distribution manager implementation in PLL*. SSC research report.
- 11 The CHIP group: *CHIP version 2.1 Reference manual*. ECRC, Arabella str. 17, D-8000 Munchen 81, West Germany.
- 12 RICHARDS, B., BETHKE, I.: *Temporal Databases: an IQ Approach*. Edinburgh University Research Paper EUCCS/RP-19. March 1988.

The 'Design to Product' Alvey Demonstrator

L.D. Burrow

GEC Electrical Projects, Boughton Road, Rugby CV21 1BU

Abstract

'Design to Product' is one of the Large Scale Demonstrators funded by the Alvey Directorate. 'Design to Product' is providing the specification for, and a partial implementation of, a system to assist designers throughout the lifecycle of light electro-mechanical products. The project includes the development of integrated flexible machining and assembly cells.

The principal feature of 'Design to Product' is its ability to integrate diverse sources of engineering information and to provide powerful techniques for using that information. The system includes a number of design support tools. These tools have use of information from the designer directly, from the evolving Product Description or from the built-in engineering knowledge bases. Information generated in the design office is used by the factory systems to manufacture the designed components.

The concepts of 'Design to Product' will be demonstrated in 1990, at the Lucas Diesel Systems factory in Gillingham. The demonstrations will be based on typical application problems and specimen parts drawn from a Lucas diesel fuel pump development and will run from early conceptual design to the machining and assembly of products.

The architecture of the 'Design to Product' system emphasises distributed computing, communications and modular construction. There is a strong relationship between the concepts of the 'Design to Product' system and the evolution of standards promoting open system CIM implementations.

1 Introduction

This paper describes the background to and the development of the 'Design to Product' Alvey Demonstrator project. It also describes the work that has been carried out and the future plans.

The Alvey Directorate was set up by the Government in 1983, with a total budget of £350 million, with the task of establishing a national programme of

collaborative research and development in IT between industry and academia. Funds for this programme were provided by the Department of Trade and Industry, the Science and Engineering Research Council and industry. Four key areas of Information Technology (IT) are supported by the programme, namely, Man-Machine Interfaces (MMI), Intelligent Knowledge Based Systems (IKBS), Software Engineering (SE) and Very Large Scale Integration (VLSI). As part of the Alvey programme a number of 'Large-Scale Demonstrators' were funded. One of these Large-Scale Demonstrators is the 'Design to Product' (DtOP) project.

'Design to Product' is concerned with applying advanced techniques in Information Technology to Computer Integrated Manufacturing. The project addresses the product lifecycle from design through manufacturing to in-service support for the class of engineering products encompassed by 'light electro-mechanical devices'. The resources available have limited the scope of 'Design to Product'. The main emphasis is towards the interactions between design and manufacture whilst simulation and analysis tools, MRP and the entire set of activities covered by Business systems are not covered (Fig. 1).

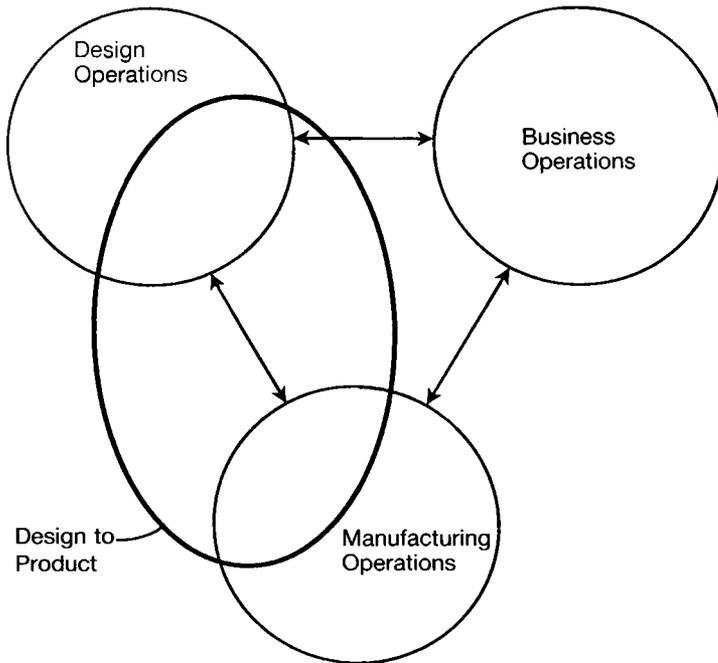


Fig. 1 Enterprise operations included in 'Design to Product'

'Design to Product' started in 1985, has a budget of nearly £9 million and a duration of 5 years. During the course of the project a little under 200 man years of effort will have been invested from both industry and academia.

The work has been divided into two 2.5 year phases. During the first 'Pilot' phase the structure of the 'Design to Product' system and the fundamental features of its key components were developed. Expertise in key Enabling Technologies was also gained, in particular by the industrial collaborators. During the current 'Implementation' phase of the project, the pilot phase software is being further developed and integrated into the full 'Design to Product' system. This DtoP system includes an example factory implementation to demonstrate the benefits of close integration between the manufacturing and design processes.

The prime objective of DtoP is to provide the specification of a system capable of assisting designers and engineers at all stages in a products lifecycle from early design through manufacturing to in-service support. A key feature of the target system is its ability to integrate diverse sources of engineering information and to provide powerful techniques for using that information. By developing a 'DtoP public interface' the system will allow third party knowledge bases and specialized processes to be included as parts of a 'Design to Product' system.

Another objective is to demonstrate a partial implementation of the 'Design to Product' system in 1990. This implementation will be based on the prototype work carried out within the project, it will also include a wide range of Alvey enabling technology projects and other research projects from both the UK and Europe. Subsequent developments will lead to commercial products in the early to mid 1990s.

To achieve these objectives a collaboration was set up between the following organizations, each concerned with particular aspects of the project. These are:

GEC Electrical Projects (responsible for overall project management, developing the integrated DtoP system, and ultimate commercial exploitation), Lucas Diesel Systems (who will stage the final demonstration, and provide the user viewpoint), GEC Marconi Research Centre (who are developing both hardware and software to support off-line programmed flexible assembly), GEC Avionics (control of the life cycle design process), Loughborough University (research and development of knowledge based process planning software), HUSAT (definition of the design process, task analysis, user interface design), Leeds University (research and development of multi-representational solid modelling systems), Edinburgh University (research and development of knowledge based integration techniques to support conceptual design).

2 Hardware and software

The design office component of 'Design to Product' is being developed on SUN/UNIX/SunNews single user workstations and IBM PCs using a variety of languages notably POPLOG (a mixed language system combining

PROLOG, LISP, POP11 and ML), C and FORTRAN. The factory area control system is being developed on DEC VAX11/750/VMS equipment and is based on INGRES. The manufacturing equipment is based around an Okuma LC20 lathe, Hulle Hille NBH70 Machining Centre, Zeiss CMM Co-ordinate Measuring Machine, and a Redifusion Reflex assembly robot (Fig. 2).

3 Design and manufacture today

Today's world of engineering design is increasingly competitive and aggressive. The complexity and performance of products is increasing yet the customer demands lower prices, higher reliability, shorter lead times and products more closely tailored to specific market requirements.

Unless industry is able to respond competitively to these pressures eventually they will be forced out of business with all the corresponding social and economic problems that this entails. Against this background the field of Computer Integrated Manufacture (CIM) is rapidly developing. However, the approach to implementing CIM has been somewhat piecemeal. Often in an attempt to resolve particular problems companies are pressurized into developing and implementing islands of automation in both Computer Aided Drafting and Computer Aided Engineering.

Throughout Europe there are a number of heavily automated manufacturing plants in existence. Generally they are purpose built, green field developments which satisfy a particular clear, generally narrow, market niche. The investment is high, the timescales well defined, and the benefits quantifiable.

However, there is another much more common environment in which CIM techniques have yet to penetrate. That is the small batch manufacturer who markets semi-bespoke products and who is effectively selling expertise in both design and the ability to manufacture efficiently. Such companies are often relatively small, cannot justify a bespoke green field development and must be able to continue to use their current resources flexibly and effectively into the future. These two contrasting environments can be shown through the differences between a plant designed for the volume production of car engines and those plants designed to produce speciality products such as prototype components for the aerospace industry.

The goal for both volume production and small batch production is the same, low unit costs, high quality and assured delivery schedules. In volume production the emphasis is towards optimising the manufacturing process to the particular product with high efficiency plant using proven concepts. Design will be strongly directed towards improving manufacturing plant efficiency. In small batch manufacture the emphasis is towards optimising the design processes, with the manufacturing operations being designed for flexibility for a range of product variants. Most small and medium enterprises have restricted resources and this implies other requirements:

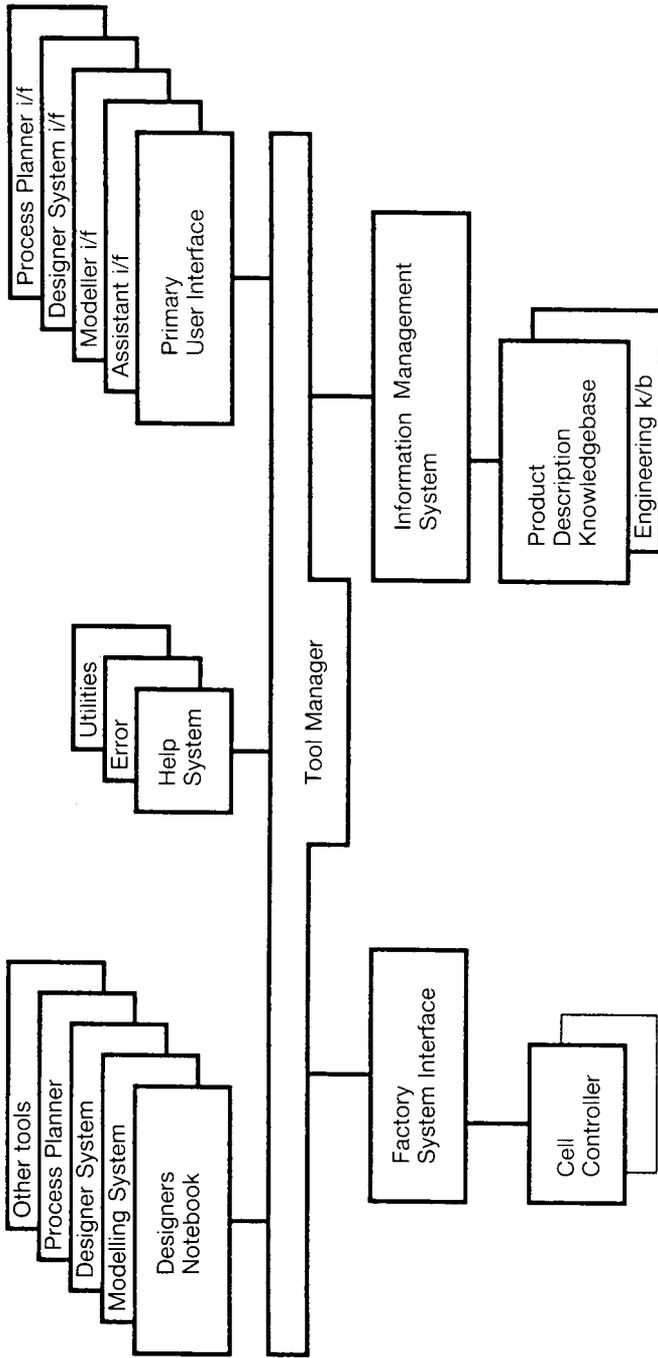


Fig. 2 The DtoP system diagram

- (a) Incremental development.
- (b) Full cost justification.
- (c) Maximal reuse of existing technology and company expertise.
- (d) Low entry cost.
- (e) 'Future safe' flexibility (implying independence of single suppliers, particular technical solutions, spreading risk).

Superimposed on this is the increasing pressure to reduce work in progress. Conventional MRP type planning approaches are seen to increase work in progress whilst JIT concepts help to reduce work in progress. The pressure is therefore to increase the proportion of a product using standard components which are pulled through on demand.

Increasingly concern is being voiced that current approaches to CIM do not offer the resilience required to allow adaptation in the future. Solutions often depend on one vendor's ability to work with another. Furthermore conventional CIM investment is seen as expensive and lacking in clear quantified cost justification.

In general terms the application of Information Technology techniques into industry has brought some benefit but not as much as had been anticipated. There are many reasons for this, amongst the more important being

- (a) The interaction between people, teams and systems is not well understood.
- (b) Information affecting important decisions is frequently not available at the right time.
- (c) There are very few integrated systems supporting an "open" range of functions relevant to design and manufacture.
- (d) System facilities that encourage learning, either of system capabilities, or from previous mistakes seldom exists.
- (e) The design cycle is often seen as a sequential process. Systems neglect the feedback and iterative processes in design leading to considerable inefficiency in the use of information.

It is against this background of design and manufacture today that the 'Design to Product' system is being developed. The key requirement is seen to be the more effective management of information; its generation, modification, retrieval and storage at all stages in the product life-cycle. The goal is to increase company effectiveness and hence profitability.

3.1 Information systems and people

The original aims of DtoP were directed towards showing that advanced software, particularly IKBS, could be used to automate the design and manufacture of light electro-mechanical items. This ambitious, technocentric view has evolved such that now DtoP is directed towards augmenting and supplementing human control rather than supplanting it.

This change in DtoP mirrors a similar change in the CIM world at large. There is now much more emphasis on understanding the underlying structures and the requirements those structures place on the information technology solutions. These developments include a much deeper appreciation of the interaction between humans and systems, enabling humans to carry out those aspects for which people are best suited whilst interworking effectively with advanced software systems.

3.2 The open system

There are two basic stances which can be adopted for CIM systems, the 'closed' homogeneous, single vendor system and the 'open' heterogeneous, multi-vendor system. From a vendor's point of view the chief requirement is to increase their market share. If the supplier is large then it may be possible to dominate a market niche by offering a widely spread set of packages integrated by an internal standard. Their recommended solution may not be the most effective or appropriate for all customers. Smaller vendors do not have sufficient resources to support such a strategy and consequently are forced to concentrate on a narrow market sector.

In contrast to this the user generally requires a bespoke system which is simple to specify and configure yet which has the potential for expansion in unforeseen directions. Frequently they will want to use equipment and software from a range of suppliers. Today the lack of interconnection standards and methods ensure that such systems are expensive to procure and few vendors are able to offer the requisite systems engineering expertise.

There is therefore an increasing number of standards bodies working to define useful, open standards for CIM. These standards cover a broad range of issues from communications (e.g. ISO OSI model, Manufacturing Automation Protocol (MAP) and Technical Office Protocol (TOP)) to data exchange formats (such as the CAM-I Product Definition Data Interface (PDDI), the ISO Standard for the Exchange of Production Information/Product Data Exchange Standard (STEP/PDES) and the Office Document Architecture (ODA) developments). Supporting these are developments concerned with systems engineering (such as the ISO Open Distributed Processing (ODP) and Portable Common Tool Environment (PCTE) [15], the US DOD CAIS and the UK Alvey work in the Advanced Networked Systems Architecture project (ANSA))¹.

However systems which support the "Open" integrated philosophy remain, at least for the time being, concepts rather than reality. In the future though, these and other initiatives directed towards establishing standards will enable high levels of integration to take place between heterogeneous systems.

From the many initiatives^{1,3,10,11} currently in place a number of themes are beginning to come together.

- (a) The need for a reference model to describe the activities of manufacturing enterprises in consistent and formal terms.
(This is required so that all parties to a CIM problem can at least agree what the problems are in terms that are not ambiguous.)
- (b) The need for standards to promote the exchange of information between systems.
- (c) The need to manage all engineering information.
- (d) The need to provide reference models for future computing systems architectures.
- (e) The need to define the role of advances in information technology notably user interface design and artificial intelligence.
- (f) The need to quantify the costs/benefits of both direct and indirect factors.
- (g) The need to adopt a 'user centred' design approach. This in turn requires the development of better techniques to effectively and accurately capture the user requirements and to transpose them into terms that system builders comprehend. The goal is to implement systems that suit the intended users.
- (h) The need for greater understanding and models describing both the partitioning of and interactions within the 'Sociotechnical' system comprising the users and the software/hardware they use.

3.3 Information supply

The processes of design, analysis, machining, build, test and quality assurance generate a considerable volume of information about the particular product; about its function, shape, manufacture and test performance. As information is encoded and transferred from stage to stage some is lost only to be reconstructed (possibly expensively and inaccurately) at some later stage, and what survives is distributed throughout the organization often in a haphazard manner. Furthermore attempts to transfer information in ways other than that envisaged initially often lead to inconsistencies and difficulties⁴.

The introduction of Computer Aided Drafting/Design and Computer Aided Engineering systems into the traditional engineering design and manufacturing procedures has not changed the fundamentally sequential process described. What they have done is to improve the efficiency and speed with which certain stages in the sequence are performed. Through their support of standard parts and part libraries, they have improved the information distribution problem in certain areas. However, the lack of good data exchange standards and lack of management of the information flow means they do little to correct the information scattering, loss and regeneration problem.

4 The processes of design

Underpinning the work of DtoP is a view of the design processes the system is intended to support. To define the specification of DtoP requires an

understanding of the basic processes the system needs to support it in order to ascertain what should be included.

There is no single generic model of design activity. Although there are a number of features that most accept will form some part of the process, the balance and dynamics between them is less clear. One such is the continuous interplay between procedural strategies and opportunistic strategies for finding design solutions.

Thus companies typically have an organisational structure representing the company's division of responsibilities. The distribution of a new project in this organisation forms the first stage of the strategy for generating design solutions. Formal company procedures may also define the information flow between the different operations in the company. However when viewed at the level of an individual the interpretation of his tasks seems highly idiosyncratic driven more by the person's style, knowledge and creativity than by predefined organisation. At this level the design process is personal and takes as input possibly fast changing information (specifications, constraints) from elsewhere. A complex iterative process, characterised by the continuous generation of new problems, review of constraints and synthesis of solutions, ultimately transforms inputs into output information to be passed to others in the system. This transformation may be seen as quasi-continuous or discrete. Project planners and organisers tend to view 'outputs' as discrete measurable deliverables although individuals may see them as the results of a continuous evolution.

Much research investigation has been devoted to the processes of design, see for example^{12,2}. Reference ¹³ describes the background of the approach adopted in the development of the full 'Design to Product' system.

5 The concept of DtoP – management of information

The goal of the 'Design to Product' system is to reduce the losses in the product information generation cycle, and to resolve the information consistency and management problems inherent in existing design and manufacturing methodologies. This is achieved by comprehensively managing the generation and use of information. There are two principal categories of information to be handled. The first is the 'Product Description' which is the knowledge describing all aspects of the product and which is built dynamically as design evolves. The second is the 'Engineering Knowledgebase'. This is the essentially static information describing the background facts and knowledge used by the designer, including for example company knowledge, product information, expertise encoded into knowledgebases and databases, and previous designs. It is the global availability of this knowledge generated and used during a product's design, manufacture and service life that is seen as the underlying unifying and integrating factor. Design is seen, not necessarily as a sequence of design, planning and manufacturing stages, but as a whole set of product generation activity interworking with a common set of information.

5.1 The DtoP system implementation

The implementation design of the DtoP system provides a central Engineering knowledge base called the 'Information Management System' (IMS) that holds both the Product Description and the Engineering Knowledge Base. The consistency of the Product Description is maintained and managed by the Information Management System. Arranged around the IMS is an extensible set of design and manufacturing support tools able to access its information. The processes of design are carried out by the user finding and selecting useful information either from the IMS or from outside of DtoP, then choosing the processes to operate on the information. Powerful facilities are provided in the user interface to the IMS to help the finding, examination and selection of appropriate information. An 'Assistant Tool' helps the user to define and/or select 'Design Procedures' and then control the processing of the selected information using the Design Procedures. A Design Procedure is any useful set of tool operations, possibly derived from company procedures, or defined by the user or other users. It provides a suggested sequence of tool operations that can be executed, suspended, modified or ignored at will by the user. By this means routine operations can be supported efficiently whilst enabling the user to explore alternatives easily. The building of the Design Procedures is one of the mechanisms available in DtoP for capturing the processes that designers use and making them available to other perhaps less skilled users.

'Design to Product' is intended to support teams of 5–20 people working on a project. Such teams are expected to use distributed networked single user workstations possibly located in different geographical locations. Such networks would also support specialised services such as database and printer servers. The system administrator has facilities to define what tools, procedures and information particular users have access to, so reducing the system complexity as seen by the user and providing a level of project management.

5.2 The architecture of the 'Design to Product' system

The 'Design to Product' system (Fig. 3) is the collection of software modules integrated into the software system that will be demonstrated in 1990. The principle requirements on the architecture of this system are that it should provide an 'open', distributed, heterogeneous, computing platform. The platform should allow functionality to be incorporated incrementally yet allow a high degree of integration. It should be scaleable, i.e. capable of long term expansion to provide greater performance and increased capabilities without running against inherent limitations of the system that are costly to overcome. The platform should be adaptable, i.e. it should allow a steady evolution as requirements and environments change.

To achieve these requirements each Tool in the 'Design to Product' system is treated as a self contained software object with a well defined interface. Tools

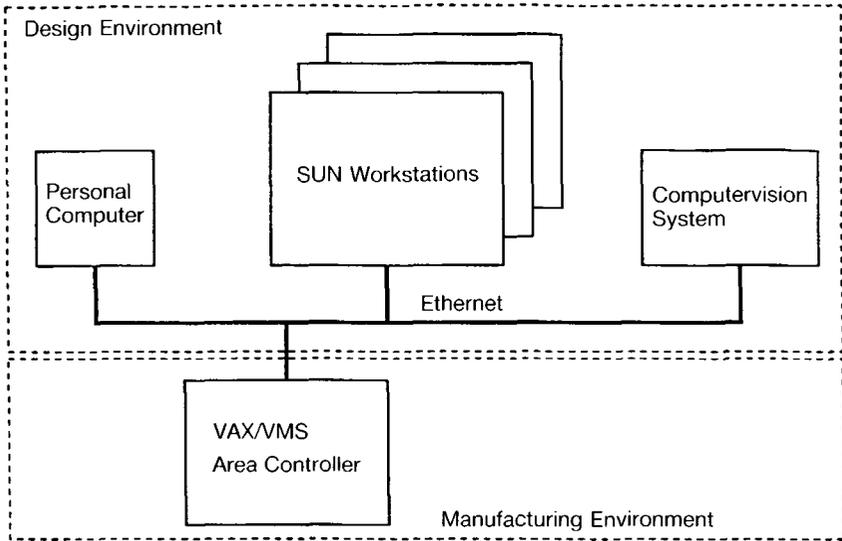


Fig. 3 Hardware configuration of the 'Design to Product' system

communicate via the Tool Manager using messages that define the actions to be executed and the information required to support the action. Tools may reside on any machine in the network, the choice of which machine would be determined by issues of performance, load sharing and local facilities.

As the design information is generated and used in the system it is recorded in the IMS as part of the 'Product Description' knowledge base. The 'Product Description' can be seen as a graph of related items of design information, dynamically constructed as the design proceeds. The graph records the dependencies between items and the processes that created the items. Its structure allows the consistency of the design to be checked and for the history of its evolution to be traced. Schema Definitions for the design items held in the IMS define the attributes of and relations between items. Where possible the representations used, particularly for manufacturing information, take account of emerging standards. This will assist the ultimate interfacing of third party systems. The comprehensive management of the design as it develops will provide designers with much better records of their work. This combined with powerful design support will allow major improvements in design productivity.

5.3 The DtoP system components

The DtoP system can be described as partitioned into a number of regions (Fig. 2).

The Design Support Tools – these are software systems designed to support particular aspects of the Design and Manufacturing lifecycle. Each of the

DtoP collaborators is developing different tools which are described briefly below. Design tools could be supplied from any source and may run in a variety of computing environments. DtoP will be demonstrating design tools based on SUN/UNIX and IBM PC/MS-DOS together with a Computer-*vision* CADD54 CAD workstation.

The User Interfaces – these are the functions that mediate between the user and the functions available in the ‘Design to Product’ system. Typically these interfaces are tailored to support particular tasks that users wish to carry out. The detail facilities available at the interfaces depends on who the user is, what their role is and what tasks they are carrying out. In all cases the interfaces are intended to give a user-friendly, convenient but flexible access to those facilities of DtoP that are particularly relevant to the goals of the user.

The Tool Manager – The Tool Manager provides the facilities for integrating the control of and data transport between a wide range of tools in a heterogeneous, distributed environment; the Tool Manager concepts are closely related to the Advanced Networked Systems Architecture (ANSA) project also supported by the Alvey Directorate¹. Its operation is largely invisible to the user but it plays a crucial role in connecting client tools to server tools and transferring information from one to the other.

The Information Management System – All the information generated and transformed by the tools of the DtoP system is managed by the facilities of the ‘Information Management System’ (IMS). The IMS provides secure, consistent and longterm storage of all information defining the Product Description. It also controls the access to the engineering knowledge stored in the ‘Design to Product’ system. This domain knowledge includes for example, records of previous designs, rulebases for expert system support, system help information, task procedures and external databases.

The lower levels of the IMS support an entity relationship model in which all objects are instances of a class type. The schema of a particular class type is defined by its position in a class hierarchy and can inherit properties from its parents. There is no particular restriction on the attribute value types. Relations can be defined either as simple links or, for more complex relations, as relation objects. One attribute of the object is the ‘contents’ of the object. This contains information that can only be interpreted by the design tools able to operate on that class of object. All entities have unique names and sets can be defined as collections of objects. Entities can be deleted, modified or created as required and found by matching attributes, and class or set membership.

The IMS has the dual roles of controlling the shared use of information and for providing long term, secure storage of important information. Thus mechanisms for archiving, consistency checking and recovery are also included. Facilities built into the IMS allow the user to specify the degree of

restriction he wishes to place on other users wishing to use or modify the same information. In the event of conflicts arising the system can resolve the priorities if required or refer to the user for arbitration.

6 The User-Computer Interfaces to the 'Design to Product' system

The principle requirement of DtoP is to produce a system that will assist the various users throughout the product generation cycle. For any DtoP type system to be successful the user must have a clear and accurate view of the system processes being supported, similarly the system designer requires a clear and accurate view of what the users are likely to require. Without these views being well understood and defined there is a real risk of developing a system in which there is a serious mismatch between what the user expects and what the system does.

The inevitable consequence of getting this aspect of the system wrong is poor performance of the combined human computer system. Training time and associated costs will be high, confidence in, and productivity from, the system will be low. 'Design to Product' has devoted considerable resources to understanding the design problem, developing methods of describing user interactions with systems, and carrying out substantial studies of the target design and manufacturing operations. This work is drawing on techniques such as the Enterprise Modelling methodology from the ESPRIT AMICE project¹⁴ and the Alvey Project SE015 'Formal Requirements Specification - FOREST'¹⁶,

To ensure that the 'Design to Product' system provides facilities that match the tasks of design and manufacture requires a firm understanding of the detail processes of design. Within the project HUSAT is studying the procedures Lucas designers currently use to design typical diesel fuel pump parts. A technique called Personal Task Representation (PTR)⁷ has been used in this study. The method principally comprises a process of structured interviews, the results of which can be configured into a structured flow chart describing the functions and information flows. The insight gained from this work has helped the development of the 'Design to Product' system by indicating what domain knowledge should be contained in the system, what tools would be useful and what characteristics the user interfaces for these tools should have.

The complexity of the design task combined with the flexibility and range of resources in the 'Design to Product' system poses many challenging problems of user interface design. The final characteristics of the user interfaces for the 'Design to Product' system have not yet been fully defined as they will depend on extensive user trials planned to take place in the middle of 1989. They will also depend upon the outcome of the current studies of the design processes at Lucas and the functional capabilities of the design support tools still under development in the project. Although these studies have yet to be completed it is apparent that the designer must be able to call up extensive

and very well developed help, support and explanation systems that inform quickly and efficiently the system state at any time. It is also clear that the system needs a good understanding of the types of user (e.g. system administrator, chief designer, apprentice), the roles they have (e.g. knowledgebase maintenance, design review, progress audit) and the tasks (e.g. editing, searching for relevant previous design information), to be carried out².

The system design of DtoP has been organised to decouple the functions required to support the user interactions (which are the design tool operations) and the detailed techniques and styles of the screen operations. Thus the user interactions with the screen are handled by specialised tools that interact with the design support tools via the Tool Manager. The user interface specifications and designs have been developed using prototyping tools and mockups. These are then implemented in a modular re-useable manner. The choice to build the user interfaces using 'SUN Network Extensible Window System (NEWS)' was made primarily because SUN-NEWS offers an object based environment ideally suited to the construction of user interfaces by the incremental development path described above.

7 Design Support Tools in DtoP

The Design Support Tools of the DtoP system assist the designer to make effective decisions, and their integration enables the design information to be shared by all tools. Such tools must be capable of working in conjunction with existing systems and techniques. The system design of DtoP provides for a 'Public Tool Interface' allowing a wide variety of tools from any source to be integrated.

7.1 Specialists

Designers frequently consult colleagues, specialist books or use standard procedures to develop the detail of some aspect of the design. Subsystems called 'Specialists' within the DtoP system will play a similar support role. Specialists could be developed that will assist the user to take account of manufacturing constraints, estimate costs, and assess the reliability and maintainability of their design. The concept of a Specialist is similar to that of an Expert System, but because many such specialists can be invoked and are positively linked to the tasks the user is carrying out considerably greater utility is achieved. Currently an advanced expert system building tool 'Socrates'⁸ is being integrated into the DtoP system. Simple PC based expert systems providing design expertise are also being integrated. Ultimately a fully developed 'Design to Product' system would incorporate many such 'Specialists' drawn from diverse sources and covering many aspects of the product life cycle.

7.2 The Edinburgh Designer System

Edinburgh University using techniques drawn from AI research have

developed an integrated suite of 'intelligent' design support tools, called the Edinburgh Designer System (EDS)⁶. A 'Blackboard' architecture has been used and a truth maintenance system based on the 'deKleer' model⁹ is supported. The system has been designed to aid the user develop the design of parts and products, starting with an initial technical specification. Engineering knowledge in the EDS is held in a Declarative Knowledge Base (DKB) as information ('Module Class Definitions (MCD)') organized in a network structure. Each Module contains declarative knowledge using formal representations developed by the project. This knowledge describes a particular engineering entity of interest. In the Lucas diesel fuel pump domain targetted by 'Design to Product' typical modules might be based on the parts of a fuel pump and contain information describing the constraints between parameters of the part, its shape, its relationships with other parts, and tabular product data relevant to the part. Alternatively the MCDs can be used to represent knowledge about regions of design space which can be explored to find a design specification that meets the requirements.

7.3 The Solid Modelling Engine (SME)

The Solid Modelling Engine in 'Design to Product' is being built at Leeds University. The SME consists of a core of multirepresentational geometric modellers; these are integrated with an 'intelligent' front end by the Process Integration and Modeller Management System (PIMMS) with a Geometry Database (GDB) system utilities and other geometric functions. The SME is seen as providing a set of modeller functions through which geometric queries may be answered. The PIMMS presents a transparent function call interface to the underlying modellers.

The SME will provide a vital system tool capable of supporting other processes in the DtoP system. Typically it will be used to calculate geometry and mass property information, evaluate dimensions, recalculate tolerance frames, generate and verify tool paths for the NC program generation.

The physical appearance of many products is an important aspect of their design. To support this the SME provides visualization capabilities typical of conventional CAD systems. An IGES interface is included in the SME to allow the transfer of geometry information to conventional draughting packages or to other modelling systems.

7.4 The Generation of Manufacturing Data

Once a design has been developed to a point that it can be described geometrically then it becomes possible to generate process plans to describe the manufacturing operations required to make the part. Conventionally a production engineer will use a 'group technology' approach to process planning. In this, parts related to the manufacturer's product range are categorized into families that are manufactured in a similar manner. A target

part is classified into one of the predefined families and its corresponding generic process plans selected. This provides the starting point for the design of a bespoke process plan for the target part.

In 'Design to Product' Loughborough University have taken the same basic approach when developing the Loughborough University Manufacturing Planner (LUMP). However, by taking a more advanced view of the issues and integrating process planning with feature visualization, automatic generation of cutter paths and AI based process planning⁵, it is believed substantial benefits in accuracy and productivity will be achieved.

One aspect of the work in the 'Generation of Manufacturing Data' section of 'Design to Product' is directed to developing knowledge bases for process planning. Another aspect is devoted to the design and construction of software tools to assist particular process planning activities, such as the selection of cutting tools and fixtures, machine tools and cutting processes and the ordering of process operations. The key characteristic of this work is that all the process planning tools can access both the Product Descriptions and the Knowledge Bases of the 'Design to Product' system. The output of the process planning activity extends the formal description of the design held in the Product Description. This integration of functions by using a common set of knowledge bases ensures that knowledge generated early in the design process is always available for use later in the process. This contrasts strongly with the current situation where the information may well not be explicitly carried forward and has to be expensively and sometimes inaccurately recreated at a later stage.

Once an ordered set of operations to manufacture the part has been decided, NC part programs can be compiled.

Many of the techniques being applied in this section of the project are known and being used today although not generally in an integrated manner. The real benefit in using formal data and methodologies in the manufacturing areas is to ensure that the design rules from which it has been derived are correct. Feedback which provides the ability to modify and improve the original data is the key difference that makes systems using an integrated approach so powerful. The emphasis is not on having the most efficient process plan or NC program, but in having an effective solution (with hopefully an alternative) quickly. Many will have experienced situations where part programmers edit part programs to bring about small time savings while machines stand idle and operators look on.

7.5 The Generation of Assembly Data

This area of work in 'Design to Product' parallels the Generation of Manufacturing Data process for automated assembly. However, the state of the art in automated assembly is not as advanced as for automated machining. There is still fundamental work to be carried out to identify the

techniques for the control of robotic assembly systems before the 'Generation of Assembly Data' processes can be clearly defined. The goal is to be able to offline programme flexible assembly cells. There are many parallels between assembly and machining which suggest that in the future the two areas (at least in part) may be supported by common tools.

7.6 Documentation support in DtoP

Much of the product design information will ultimately be contained in documentation such as product design descriptions, publicity material, drawings, operator instructions, service and maintenance manuals. Accordingly DtoP includes some basic documentation support tools. The documentation model supported in DtoP consists of a generic structure description populated by text or diagram elements. These text or diagram elements are taken from the Product Description and are created automatically by some tools, or input by the Designer in the course of his work. The Designers Notebook and the graphic support tools in DtoP allow the designer to produce text or diagram notes at any stage of the design and to tag them to the Product Description. The document descriptions are postprocessed for display on the screen, for output to a printer or for input into wordprocessing or desk top publishing packages.

7.7 The Factory Control System

Within the 'Design to Product' system the factory control system will not be deeply integrated into the business system which typically controls the manufacturing enterprise. However, during the final demonstration illustrative links between the factory control system and a business system will be demonstrated.

The development of the factory systems for the project is also in two phases, pilot and implementation. For the pilot phase a small FMS machining cell was constructed by GEC – Electrical Projects 'FAST' division at Rugby. The cell included a CNC lathe, CNC machining centre with robot loading, a CNC inspection machine and the GEC free ranging AGV. The whole is controlled by an advanced modular cell controller developed at Rugby. Process plans and part programs generated by the 'Design to Product' system can be accessed by the cell controller. Part program development can then be carried out on the shop floor and returned to the 'Design to Product' system to form part of the evolving Product Description. By this means a consistent record of the design and machining evolution can be maintained. This cell was demonstrated publicly early in 1988.

A prototype flexible assembly cell is being developed by GEC – Marconi Research Centre at Gt. Baddow. This is based around the TETRABOT (a parallel robot being developed by GEC). This cell is being used to investigate assembly techniques using real-time sensor feedback and reconfigurable grippers.

Running in parallel with the GEC Marconi Research work is the Lucas Research assembly cell. Lucas are building an assembly cell based on current technology directed towards analysing and developing appropriate assembly techniques for flexible manufacture. This cell along with an Okuma LC20 lathe, Hulle Hille NBH70 Machining Centre, Zeiss CMM Co-ordinate Measuring Machine, will form the manufacturing part of the DtoP demonstration 1990.

7.8 Other Design Support Tools

Tools to support the analysis of parts (e.g. for strength, vibration and thermal characteristics) or to simulate their operation are not being developed as part of the project. However, the DtoP system is being developed to include a public tool interface and therefore such tools are likely to be components of future fully developed systems.

7 Future prospects

The Alvey Large Scale Demonstrator projects are intended both to show possible applications of research in three key areas (MMI, IKBS, SE) and to accelerate the introduction of advanced products to the market place. 'Design to Product' will demonstrate in 1990 certain aspects of its concept in detail. Thereafter development of commercial products will continue. To be commercially valuable a 'Design to Product' system must be a tool that designers both wish to use and find helps them (in a cost effective manner) to produce better solutions to their design problems. To achieve the first the system must have effective user interfaces, provide useful design tools and have a good performance. To achieve the second requires an extensive period of knowledge engineering to create the general knowledge bases contained in the system that describe the application design environment to be supported. Throughout the life of the system these knowledge bases will have to be updated and extended. Tools to support this maintenance will be part of the system. Finally to be acceptable commercially the system must be able to work with the design tools customers currently use. Consequently the 'Design to Product' system must provide interfaces using data exchange standards where they exist.

8 Conclusions

'Design to Product' offers the possibility of a novel and integrated approach to design and manufacture. It will provide specialized tools to support important parts of the design process, provide the mechanisms for integrating the tools and manage the records of the design description as they evolve.

By providing an integrated and direct path to the factory systems it should prove possible to move rapidly and accurately from the design phase of a product to its manufacture.

Finally as more standards evolve which support open systems then many of the 'Design to Product' system concepts described in this paper will extend beyond the bounds of this project.

Acknowledgements

This paper draws extensively on the research and development work of all the collaborators in the Alvey 'Design to Product' project. The 'Design to Product' project is supported by the Alvey Directorate (Information Engineering Directorate) The Science and Engineering Research Council and the Department of Trade and Industry.

References

- 1 ANSA Reference Manual Release 00.03, ANSA 24 Hills Rd, Cambridge UK.
- 2 Alvey Project MMI142 – A User Modelling Tool for MMI Design, Project Report (unpublished).
- 3 ESPRIT I, Area 5 Project 688 "AMICE". Paper "CIM-OSA: The Enterprise Model", CIM Europe Conference, Madrid, May 1988.
- 4 SMITHERS, T.: "The Alvey Large Scale Demonstrator Project Design to Product" in Bernold, T. (Ed.), "Artificial Intelligence in Manufacturing", Elsevier Science Publishers B.V. (North Holland), 1987.
- 5 Process Planning using a Truth Maintenance System. UMIST/ACME workshop and advanced research in CAM, January 1987.
- 6 POPPLESTON, R., SMITHERS, T., CORNEY, J., KOUTSOU, A., MILLINGTON, K. and SAHAR, S.: "Engineering Design Support Systems", presented at the 1st Int. Conf. Applications of Artificial Intelligence to Engineering Problems", Southampton, April 1986.
- 7 "The Navy Engineering Standard", Procurement Executive, Ministry of Defence.
- 8 CORLETT, R., DAVIES, N., KHAN, R., REICHGELT, H., VAN HARMELEN, F.: "Socrates: A flexible toolkit for building logic-based expert systems", in Knowledge Based Systems, Vol. 1, No. 3, June 1988.
- 9 DE KLEER, J.: "An Assumption Based Truth Maintenance System", in Artificial Intelligence 28, 1986, pp. 127-162.
- 10 SULLIVAN, J.S., RUMMIER, D.C.: "Second generation CAE tools learn to share one database", Electronic Design, July 10, 1986.
- 11 MURPHY, S.: "Human Centred CIM Systems", ESPRIT I Project No. 1199, Proceedings of the 5th Annual ESPRIT Conference, Brussels, November 14-17, 1988.
- 12 FINKELSTEIN, L., FINKELSTEIN, A.: "Review of Design Methodology", Proceedings of the IEE, Vol. 130, Pt. A, No. 4, June 1983.
- 13 SINCLAIR, M., SIEMIENIUCH, C., JOHN, P.: "A User Centred Approach to Define High Level Requirements for Next Generation CAD Systems for Mechanical Engineering", DtoP Project Document DTOP/PROJ/HUSAT/26/1, (to be published).
- 14 ESPRIT Project No. 688 AMICE, CIM-OSA Reference Architecture Specification, available from CIM-OSA/AMICE, 489 Avenue Louise, B 14-B-1050 Brussels, Belgium.
- 15 ESPRIT Project No. 32 PCTE, The PCTE Functional Specifications Manual.
- 16 FINKELSTEIN A, POTTS, C.: "Structured Common Sense: A Requirements Elicitation and Formalisation Method for Modal Action Logic", ALVEY Project No. SE 015 FOREST, Deliverable Report 2, November 1986.

Notes on Authors

E. Babb

Ed Babb obtained qualifications in Electrical Engineering and Computer Science at Imperial College. He then researched adaptive pattern recognition systems at Cambridge University on secondment from Hawker Siddely Dynamics. From about 1971, working in ICL, he researched speak recognition and information retrieval. His work on CAFS covered storage structures, architectures and query languages. He is now studying the application of mathematical logic to business and is currently Manager of the Logic Language Project in the Systems Strategy Centre at Bracknell.

Dr. L.D. Burrow

After completing a doctorate at Warwick University Laurie Burrow spent three years engaged in research in the automation of urban transport systems. He joined GEC Electrical Projects as a systems engineer concerned with the design, development and implementation of control systems for projects in the metals and marine industries. From 1981 to 1984 he worked for the Kenya Government, setting up the Control and Instrumentation courses and facilities at Mombasa Polytechnic. In 1984 he rejoined GEC Electrical Projects as a Technical Manager of the Design to Product Alvey Large Scale Demonstrator.

C.P. Burton

Chris Burton joined Ferranti in 1957 after graduating in Electrical Engineering at Birmingham and two years' square-bashing and maintaining army radar. Helped commission and then maintained a large valve computer system for three years, then moved to development of Orion and the FP6000/1900 series at West Gorton prior to Ferranti Computers becoming part of ICT (later ICL). Managed the hardware development teams for the planned 1908A and the original 2970 and 2980 mainframes. Participated in the mainframe development of the original S-series 2900s, and in 1977 transferred to the newly formed Advanced Development Group, mostly investigating the then new microprocessors and networks. This eventually became part of the Knowledge Engineering Business Centre, where he initially led the team introducing AI tools such as LISP and PROLOG. He is currently a member of the Alvey/DHSS Large Demonstrator Project, with interests in 'Large Scale' aspects such as conversion and analysis of source material and management of the knowledge base. He is a Fellow of the IEE and of the BCS.

M.R. Gunner

Mike Gunner graduated from University College London with a first degree in physics. For eleven years he taught physics in schools in England and Sierra Leone and in the University of Zululand, South Africa. He has a Post-Graduate Certificate in Education and a Diploma in Education, both from London University. Since 1977 he has held a series of appointments in ICL, in training, technical support, software development and validation, and consultancy.

From 1986 until 1988 he studied for an MSc in Information Systems Design at Kingston Polytechnic, receiving sponsorship from ICL; the work described in this paper was completed as a project for the MSc. He has recently become a consultant in the Services, Application and Systems Support Group in the ICL(UK) Central Government Business Unit.

Professor P. Henderson

Peter Henderson is Professor of Computer Science in the Department of Electronics and Computer Science at the University of Southampton. Previously he was Professor of Information Technology at the University of Stirling and before that a lecturer at the Universities of Newcastle upon Tyne and Oxford. He is the author of a well-known book on functional programming and numerous papers on software engineering. He has been a consultant to ICL for many years.

Dr. V.M. Jones

Valerie Jones graduated in English Language and Literature from the University of Newcastle upon Tyne and in 1979 was awarded a PhD by Newcastle for a thesis on "Some problems in the computation of sociolinguistic data". She then held posts at Newcastle, first as a Research Associate in the Department of English Language and then as a Programming Advisor in the Computing Laboratory. In 1984 she moved to the Department of Computing Science at the University of Stirling where she is now a Research Fellow working on the ESPRIT-II Lotosphere Project. Her current interests are in software engineering, formal methods, methods for rapid prototyping and expert systems. She is either sole or joint author of some 20 publications, roughly half on topics in linguistics and half (the more recent) on topics in computing science; her second book, written jointly with Dr. Heather Alexander (until recently at Stirling), on Software Design and Prototyping using *me too*, will be published during 1989.

R. Lucas

Roger Lucas is the Manager of Strategy Integration in the Systems Integration Strategy unit within ICL Product Operations HQ. He joined ICL – well, English Electric Leo Marconi – in 1965 as a programmer, later progressing through customer support on System 4 and 1900 to Operating System technical support and consultancy, mainly on George 2, VME/B and

VME/K. He was the ICL representative on the VME/K User Group throughout its life cycle.

In more recent times he has worked in ICL Services on Information Systems strategy and implementation for the Customer Service organisation. For the last 18 months he has worked in his current role in Systems Integration Strategy, initially within Marketing and Technical Strategy and now in Product Operations.

Y. Pisigot

Yannick Pisigot joined ICL in November 1986. After having spent several years in the computing divisions of industrial companies (Philips, Electrolux) he gained important retail experience while setting up the management of the computing division of a hypermarket chain (Continental). His first mission with ICL was to create a development and support team in the newly-established vertical retail activity. He is currently responsible for the Retail products and solutions team in ICL France.

Dr. J. Michael Quinn

He joined ICL Marketing and Technical Support in 1969. Since then he has held a number of positions in planning, project management and software development in various corners of the UK. He is currently a member of the European Declarative System (Esprit II) project team in Mainframe Technology Centre at ICL West Gorton. He became interested in geographic information systems on joining Futures Group some six years ago; the work reported here stems from that time and has been carried out in co-operation with the PLANES project based at Leeds.

A.J. Russell

Fred Russell joined ICL in 1967 as a Systems Engineer, subsequently transferring to selling 1900 systems in the National Accounts Area. On leaving the sales force he had a varied career within ICL, ranging through business planning, market introduction and programme management of 2900 development and introduction. He then became involved in the emerging office automation strategy, in particular in the human factors aspect of this. Currently he manages the Human Factors Group in the Systems Strategy Centre at Bracknell and is responsible for the ICL component of an ESPRIT programme aimed at developing knowledge-based user-centred software development (CASE) tooling.

Dr. R.A. Snowdon

Bob Snowdon received his PhD in 1974 from the University of Newcastle upon Tyne, having studied programming methodology and how computers can be used to assist in the task of software system design. He joined ICL in 1975 as one of the designers of the 2900 CADES system. More recently he has been closely involved with both UK and European initiatives in software

engineering, particularly in the areas of formal development methods and computer systems to support the many aspects of software systems development. Since 1985 he has been the project architect of the Alvey-funded IPSE 2.5 project.

Dr. G.E. Storrs

Graham Storrs is a senior consultant with Logica Cambridge Ltd., the research and development centre for Logica worldwide. He works within the Human-Computer Interaction Group, which is responsible for a range of products in the areas of Intelligent Training, Intelligent Front-Ends and HCI design methods. His present activities include leading a design team in the Alvey DHSS Demonstrator Project and contributing to a study into support for the development and dissemination of methods for the CCTA. Previous work in Logica has included the development of intelligent training systems.

His background has been in human factors consultancy and in academic research into HCI and artificial intelligence. His PhD work was in the field of cognitive psychology. He is a founder and editor of the international HCI journal, *Interacting with Computers*.

D.E. Talbot

David Talbot joined ICL in 1960 after graduating from Oxford with a degree in mathematics. He has held a wide variety of jobs: a systems analyst/technical manager; a business manager responsible for ICL's business with UK Ministry of Defence research establishments and with Universities; a marketing manager for ICL(UK) and for ICL's mainframes worldwide.

Most recently he completed a secondment to DTI where he was responsible for the Software Engineering aspects of the Alvey Programme. He is currently Manager of Technology and Engineering in ICL Product Operations. Throughout his career he has had an interest in the design, development and application of systems "in the large".

P.W. Veasey

Philip Veasey received an M.Sc in Mathematics at the University of Warwick in 1968. His experience has included seven years selling computer systems with Burroughs and Microdata and five years teaching in Third-World universities. Before joining ICL in February 1985 he was responsible for marketing project management software in South East Asia. This experience led to the strong planning orientation which has influenced his work as architect of the SENSE methodology now used by STC Corporate Information Systems for all internal systems development. He is now programme manager for SENSE IPSE, which will implement SENSE on the IPSE 2.5 technology.

Professor B.C. Warboys

Brian Warboys has worked for ICL (and its predecessors) since graduating with a degree in mathematics at Southampton University. He has specialised in the development of operating systems for mainframes: throughout the 1970s he was Chief Designer of the VME Operating Systems for the ICL 2900 series machines, and as such was responsible for both the product design and development of the CADES systems used to aid the development process.

He is currently both Professor of Software Engineering in the Computer Science Department at Manchester University and an ICL Fellow. At Manchester he heads a team of 6 Research Assistants and 4 Ph.D students concerned with the development of a Software Environment for the Flagship machine, an Alvey parallel machine project being developed in collaboration with ICL Mainframe Systems.

E. Wilson

Ed Wilson joined Inland Revenue from school in 1962, entering Automatic Data Processing (ADP) in 1965. Whilst still employed by IR he had wider experience through loans to obtain a Master's degree in Design of Information Systems at the Royal Military College of Science, Shrivenham and latterly collaborating on the Heineken Project with ICL West Gorton. This was a joint HM Government/ICL venture to validate and advance the in-service date of Series 39, particularly Level 80, by 12 months.

Once this project had enabled IR to successfully intercept Series 39 he returned full time to the Revenue and in January 1988 was appointed head of Technology Group at Telford, responsible for some 32,000 VDUs and 50 Series 39 Level 80 nodes. He now has additional responsibility as Technical Architect for the whole of IR data processing at Telford, Basingstoke and Worthing.

