ICL TECHNICAL JOURNAL

Volume 6 Issue 4 November 1989

Published by
INTERNATIONAL COMPUTERS LIMITED
at
OXFORD UNIVERSITY PRESS

ICL TECHNICAL JOURNAL

The ICL Technical Journal is published twice a year by International Computers Limited at Oxford University Press.

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

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, Pinkhill House, Southfield Road, Eynsham, Oxford OX8 1JJ.

Printed by H Charlesworth & Co Ltd, Huddersfield ISSN 0142–1557

Contents

ICL TECHNICAL JOURNAL
Volume 6 Issue 4

Valedictory Editorial

This is the last issue of the ICL Technical Journal that I shall edit; I am handing over this privilege to my very good friend and colleague, Dr. John Pinkerton – who was much involved in the setting up of the journal and has been a member of the Editorial Board from the start. Both of us have been concerned with the electronic digital computer from its very first days but he has the distinction of having designed one of the truly pioneering machines, LEO – "Lyons Electronic Office". Based on the design of the Cambridge University EDSAC, this was doing routine clerical work for the catering company J. Lyons & Co. in 1951 and has a good claim to be the world's first business data processing machine: at the very least it was among the first.

The first issue of the Technical Journal was that of November 1978 (this, November 1989, is the 23rd); that was 11 years ago, which is a long time in the history of the computer. Two developments have made the computer world of today a very different one from that of 1978: the scarcely believable advances in circuit and storage technologies and the greatly increased understanding of the formal properties of information. The sequence of issues reflects this. The terms VLSI (Very Large Scale Integration) and RISC (Reduced Instruction Set Computer), for example, do not appear at all in the early issues – the technologies they describe did not exist; now it is assumed that everyone knows what they mean. "Expert Systems" – which involve formal representation and manipulation of knowledge – were something of an adventure in the 1970s (the first paper we published was in November 1980); recent papers have described systems used routinely in ICL for configuring complex computer installations and planning the supply of materials to its manufacturing plants. One interesting observation is that the term Information Technology does not appear in any issue before May 1982 (Volume 3, No. 1) where we record the launching of Information Technology Year 1982 ("IT 82") by the then Minister for Information Technology, Kenneth Baker. The very next issue, by the way, has a paper on "The advance of Information Technology" by J.M.M. Pinkerton.

The journal was set up with a very broad remit; the Notes for Authors in the first issue say that the content "will have some relevance to ICL's business and will be aimed at the technical community and ICL's users and customers." In 1978 the computer – the technological base on which ICL's business rests – had already penetrated deeply into industrial, commercial and even ordinary daily life over a very wide area; that penetration is now even deeper and over an even wider area – think of supermarket check-out

tills, for example (very much ICL territory) – so it is not surprising that the Journal's coverage has increased. An indication of this is the physical size: we – meaning the Editorial Board and myself – started with the idea of each issue having about 6 papers and filling about 100 pages; these values have been about 12 and 200 respectively for the last half-dozen or so issues – a size which I (and I think my colleagues too) consider the maximum for comfortable handling.

Any regularly-published series, on whatever subject, is something of a record of history, and this is certainly so for our journal. One thing that I have found particularly interesting in looking through the past issues is that some of the problems discussed in the earliest papers are still of major importance – of greater importance than ever, in fact, because the tremendous developments in technology have made so many more applications both physically possible and economically feasible. I am thinking in particular of the problems of networking, of distributed systems and of security of electronically stored, processed and transmitted information, treated in many papers throughout the life of the journal. As the applications and the physical devices that execute them become ever more sophisticated, and the systems that are built and operated become of ever larger scale, the problems they raise become not only of great importance but also of great intellectual difficulty; these are problems that face everyone, not only ICL, and the work of the authors who have written on these problems has contributed significantly to their understanding and, because of that, to their solution.

I have just said that the journal was given a very broad remit; it has also been given a very free hand, for which the company has my sincere gratitude and respect. There was a positive decision that there would be an Editorial Board and that this should include non-ICL members, and that whilst the majority of papers would be written by ICL staff, contributions from outside ICL – in particular, from the academic world – should be sought as a matter of course: all this to reduce the risks of too much inward looking and so help to maintain standards. It has always been one of the Board's basic principles that every paper we publish must convey some real information about the subject or problem that has some significance in the real world. Some of the subjects and problems discussed are intrinsically difficult and inevitably only readers with a live interest, and probably specialist knowledge, will follow the arguments of the corresponding papers; but even here we ask the author to indicate, in the Abstract or the introductory paragraph, why the subject is important – and therefore why the paper has been written.

I have found producing this journal immensely enjoyable and stimulating, not least because it has brought me into contact with so many able, enthusiastic and thoroughly likeable people. I am not yet severing contact completely, because I shall continue as a member of the Editorial Board and collaborate closely with Dr. Pinkerton in the production of the journal. Let me record my gratitude to all the authors (something like 150, all told): writing, I know, is very hard work and, after all, without their efforts there

would be no journal; to my Editorial Board, for their help, advice and support and especially for the lively and always constructive discussions at our meetings; to our publishers, first Peter Peregrinus and then Oxford University Press, with whom it has always been a pleasure to work; and to the ICL management for allocating the funds that support the journal and for the great freedom we have been given in producing it.

Jack Howlett

"TIME TO MARKET"

The ICL University Research Council organised a Workshop on the theme of *Time to Market*, at Wokefield Park, Berkshire on 5–6 March 1989, in which 23 ICL staff and 20 members of British, continental European and American academic institutions participated. The Workshop was opened by John Dickson, Managing Director, ICL Product Operations; the principal speaker was Professor Steven Wheelwright, Harvard University, USA; and five "Syndicate" discussions were held in parallel.

There follows a Foreword by John Dickson, summarising his opening address; a paper by David Saxl of ICL Manufacturing and Logistics, dealing with the Syndicate discussions, and a paper by Professor Wheelwright, based on his address to the plenary session.

Foreword

ICL's mission is to provide high value solutions through the use of IT for its customers' problems. The core hardware and software products underpinning these solutions are produced by Product Operations. This group employs some 7000 people of whom half are in manufacturing. The value, at cost, of its annual output is $500M and is made up of some 7000 different customer-recognisable products.

Approximately 10% of the Company's revenue is spent on development funding; within Product Operations £12–15M is spent on IT and IT applications.

Time to Market – the interval between conceiving a new product and getting it on to the market – is taken very seriously. ICL, particularly in the 1970s and early 1980s, did not give enough attention to this subject: but that has now changed.

A fundamental aspect of the improved and still improving performance has been to cultivate throughout the organisation the belief that everyone, in whatever job or grade, is a member of the team and plays a part in achieving the goals of the best levels of competitive quality and reliability for each and every product; of 100% on-time delivery; of continuous inventory reduction; and – the theme of the Workshop – of the reduction of time in all activities, particularly in delivery lead time. We are introducing measures to recognise and reward such achievements. The Company is now performing at a very high level relative to its competitors; in particular, inventory turns are among the highest in the industry, as is the overall level of return on investment.

However, there is still much to be done. Design cycles are too long: mainframes take approximately four years – can we find ways of reducing this to three, and then to two? As well as developing its own programs to achieve these goals Product Operations is looking to work with academia to provide the necessary innovative thought and approaches.

John Dickson
Managing Director, ICL Product Operations

Time to Market in New Product Development[1]

Steven C. Wheelwright
Graduate School of Business Administration, Harvard University,
Boston, Mass. 02163, USA

Abstract

Increasing rates of technology development, rising expectations in the market place, and increasingly global competitors have led to shorter product life cycles in a number of industries. There are a number of consequences for manufacturing firms as a result of pressures for reduced time to market. Following an introductory section, this paper explores the gap between the promise and reality of new product development in all too many manufacturing environments. The body of the paper then introduces a number of concepts, techniques, and approaches that recent studies suggest offer considerable potential to managers seeking to improve significantly their time to market. Because of the importance of projects in new product development, the article is organized around three phases in project management: those issues that can be addressed before the project is formally staffed, approved, and initiated (pre-project activities); those issues that arise during the life of the project team and its development effort (project management); and those issues associated with how an organization learns across projects and applies that learning to improve its product development capabilities (organizational learning).

I Introduction and Perspective

Time-based competition and new product development have become hot topics both for academics and managers. Recent work in a number of industries – ranging from personal computers (and their peripherals) to automotive and from medical instruments to consumer appliances and on to manufacturing equipment – suggests at least three reasons for this surge in popularity. Each of these reflects a somewhat different perspective, and thus identifies different issues and concerns in the broad area of shortening time to market for new products.

---

[1]This paper is an edited version of a talk given by the author at the ICL University Research Council Manufacturing Workshop, held on 6 March 1989, at Wokefield Park, Mortimer, Reading, Berks. It draws on the author's prior work as well as that of his colleagues. The author would like to recognize especially the contributions of Professors Kim Clark, Robert Hayes, and Earl Sasser of the Harvard Business School in the development of many of the ideas, concepts, and examples reported in this article.

The first of these perspectives is that of the firm or business unit. For these organizations, fast product development can provide benefits related to market position, resource utilization, and organizational renewal. In terms of *market success*, rapid new product development is considered a means to get a jump on the competition, strengthen existing advantages by creating stronger competitive barriers, establish a leadership image that translates into product dominance through the setting of industry standards, access new markets and new customer segments, and enhance existing product offerings.

From the firm's perspective, anticipated benefits in *resource utilization* include capitalizing on prior R&D investments (applying discoveries made in the laboratory), improving the return on existing assets (such as the sales force, the factories, and the field service network) by giving them better products to sell, applying new technologies in both products and manufacturing processes, and eliminating or overcoming past weaknesses that have prevented other products from reaching their full potential.

Perhaps the most exciting benefits from fast product development arise in the form of prospects for *renewing and transforming* the organization. The excitement, image, and growth associated with product development efforts capture the commitment, innovation, and creativity of all parts of the organization. This success can, in turn, enhance the firm's ability to recruit the best people, improve their integration, and accelerate the pace of change and progress throughout the organization. The development projects themselves often can be the vehicle by which new approaches and new thinking are adopted and become institutional reality.

A second important perspective leading to interest in time to market and new product development topics is today's competitive arena. As witnessed in a variety of industries, the organization that can bring out new products significantly *faster* than its competitors and then repeat that process *more frequently* than competitors can develop a substantial advantage in the market place. Such organizations have the luxury of choosing between preempting competitors by introducing early, or waiting for technologies, markets, and distribution channels to shake out and then developing new products that are better targeted for tomorrow's environment.

Numerous studies indicate that CEOs and managers see the capability of rapid product development as one of the last frontiers of new competitive advantage. In addition, it is an advantage that, if not developed before competitors, will quickly become a disadvantage as one becomes a follower, finding it increasingly difficult to keep up with competitors' moves. A firm with a significant advantage in rapid product development can dictate the terms of competition to others in its industry and thus "take charge" of its environment.

A third perspective on the importance of these issues is a national one. Countries that have historically been innovators in technology and R&D

increasingly find their returns on such investments taken away by developing or recently developed countries who have better skills for applying those technologies in new products and services. For example, the inability of the U.S. with all of its investment in R&D to dominate a number of global industries where its domestic market is substantial, is of great concern today. It focuses attention on educational issues, tax laws, and trade policies.

Whatever the perspective, it is clear that time-based competition and new product development are areas of concern to a broad segment of the population. They also are likely to remain concerns for several years to come because their potential benefits can be substantial, as can their failures.

II The Reality Versus the Promise

In spite of the potential promise and perceived need for faster new product development, the reality in most firms is that such efforts fall far short of their possibilities. In many firms, the majority of product development projects fall far short of their original objectives, with development efforts proving to be painful and frustrating both to organizations and individuals. To understand what causes the disparity between this promise and reality, and, more importantly, to be in a position to take corrective action, it is useful to consider half a dozen types of commonly encountered problems. These appear to be "classics"[2].

1 *The moving target.* Too often, the basic product concept and its match with the technology and the market shift over time. This can be caused by external problems: for example, locking into a technology before it is sufficiently stable, targeting a market that suddenly changes, or making assumptions about the distribution channel that don't hold. In each of these cases, the project gets in trouble because of inadequate consistency of focus throughout its duration and an eventual mismatch with reality.

   Once the target starts to shift, the project lengthens and longer projects invariably drift more and become more of a moving target than shorter ones. Thus the problem compounds itself. Dramatic examples of such mismatches include the Ford Edsel in the mid-1950s and Texas Instruments' home computer during the late 1970s. Even very successful products like the Apple Macintosh can experience a rocky beginning and require iteration to reach an appropriate focus and positioning because of such moving target difficulties.

2 *Mismatches between functions.* While the moving target problem usually reflects a mismatch between an organization and its external environment, mismatches also often occur within an organization. What one part of the organization expects or imagines that another part can deliver may prove to be unrealistic or even impossible. For instance, engineering may design a product that its factories cannot produce – at least, not

[2] See "The New Product Development Map" by Steven C. Wheelwright and W. Earl Sasser, Jr., *Harvard Business Review*, May–June 1989, pp. 112–125.

consistently, at low cost, and with high quality. Similarly, engineering may design features into the product that marketing's established distribution channels and selling approach cannot utilize fully or existing customers don't need or are unwilling to pay for. Or, manufacturing may assume a certain mix of new products in planning its requirements while marketing makes different assumptions, confident that manufacturing can alter its mix dramatically on short notice when in fact it cannot.

Such mismatches may result from a lack of communication among the functions, or from a sequential, over-the-wall approach to development project management. Whatever the case, new product development is hampered. One of the most startling mismatches we've encountered was at an aerospace firm where manufacturing built an airframe assembly plant using one set of new product specs, only to find that the building was too small to accommodate the wing span of the aircraft as finally designed.

3 *Lack of product distinctiveness.* Often, new product development terminates in disappointment because the new product is not as unique or defensible as the organization had imagined. The problem is not that the concept was poorly executed, but that it was not nearly as robust and distinctive as the organization had imagined. This can occur because (a) early on, the organization fails to consider a full range of alternative ways to meet the set of customer needs, (b) the organization locks in on the initial concept too quickly, without thorough analysis, or (c) the market place and technology were not yet ready for the concept as incorporated in the product.

As a result, competitors are able to block or parry the new offering with their existing (or slightly modified) products. A unique example of this is Plus Development's HardCard®, a hard disk designed for easy installation into the expansion slot of a personal computer. After eighteen months of development effort, the company thought it had a unique product with at least a nine-month market lead on competitors. However, by the fifth day of the industry show at which the HardCard® was introduced, a competitor was showing a breadboard of a competing product. Within three months, that competitor was shipping product.

4 *Unexpected technical problems.* Delays, cost overruns, and feature shortcomings often can be traced to unexpected technical difficulties. There tend to be two primary causes of such technical problems: the firm overestimates its technical capabilities and discovers, belatedly, it lacks the depth and resources to do what the new product concept requires; or it mixes invention with the application of new technology during the new product development effort.

If an essential invention is not completed before the product development project starts, the entire project tends to suffer the delays, mid-course corrections, and backtracking typical of the inventing process. An industrial controls firm encountered both problems when it changed a part from metal to plastic, and discovered that its own manufacturing

processes could not hold the required tolerances and its new supplier could not provide raw material of consistent quality.

5 *Problem solving delays.* Every new product development activity involves uncertainty, both with regard to the specific problems and conflicts that will inevitably arise, and the resources required to resolve those. Too often, organizations allocate all of their development resources to known project requirements and leave little or no cushion for the unexpected. Subsequently, when the inevitable occurs – an unanticipated problem is encountered and the project experiences delay – they rob Peter to pay Paul. This siphoning of resources cascades into delays on other projects.

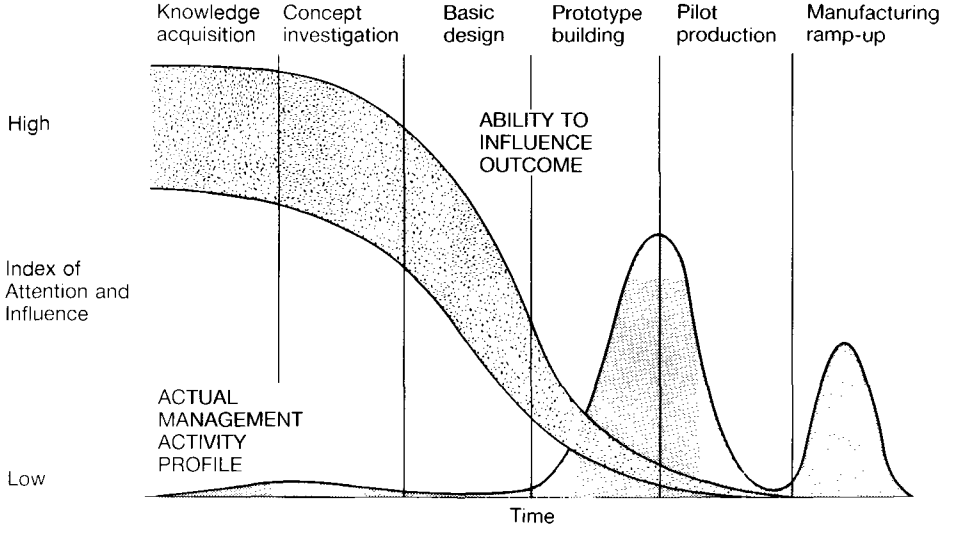
   Once delays occur, costs increase, pressures amount to cut corners, and further problems arise. When a major project got into trouble at one electronics company, key people were pulled off other projects. The company discovered that the reassigned people took weeks to get up to speed, and the project was almost as late as it would have been without them. In addition, several other projects were delayed and saw their costs escalate as well.

6 *Unresolved policy issues.* A number of very specific choices and decisions must be made during any product development project. If major policies have not been articulated clearly and shared, these project choices often force the related policy issue for the entire organization. While such forcing is not inherently bad, it inevitably involves more senior levels of management in resolving a specific issue.

   Resolving policy issues during the "heat of battle" and at senior (more politically oriented) levels of the organization inevitably engenders delay and further complications. One automotive firm that lacked a clear policy on make versus buy-in component sourcing changed the manufacturing location for a new product four times – from a headquarters plant, to offshore Mexico, to offshore Japan, to a local subcontractor – before actual start-up. Each change entailed a couple months of delay and costly design modifications.

While most firms can recite their own litany of horror stories regarding development projects that have fallen short of their original objectives and potential, management's concern is with how to avoid such problems and, more importantly, how to turn product development capability into a strong competitive advantage. Several firms have found a useful framework for organizing their thinking to be one that divides development efforts into three time-related segments. The first segment addresses actions and behaviours that occur before the formal project is set up and approved. The second considers what goes on from the start of the formal project until its completion. The third considers activities and actions that occur after the project is completed and impact the retention of lessons learned, their transfer to other development efforts, and the general enhancement of the firm's capabilities in product development. The remainder of this article samples a number of important concepts, techniques, and approaches that can be used in each of these three segments.

In today's competitive environment, it is not enough for an organization to achieve either low cost, high quality, or short development cycles. They must achieve all three. However, most firms who pursue improvements on all three fronts do so primarily in the middle segment of the development cycle – the formal project management phase. Unfortunately, management's ability to influence and shape the success of the project is already largely determined by that point. Separating the planning of a project before it begins from the project itself, and spending adequate time, effort, and resources on pre-project work, can have a substantial impact on the outcome. One of the best graphical illustrations of this is shown in Fig. 1. The data for this representation comes from a major U.S. auto company, and was collected by the consulting firm of McKinsey during their study of several new car development efforts.



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 279.

Fig. 1 Timing and Impact of Management Attention and Influence

As indicated in Fig. 1, the ability of management to influence the outcome of a project is highest in the pre-project phase. By the time prototypes are being developed, that ability is declining rapidly. However, as McKinsey found in this automotive firm, the actual profile of management activity tends to be minor in the pre-project and initial project phases and does not become substantial until the prototyping, pilot production, and manufacturing ramp-up stages of the project. This is the point at which new product programs tend to run into difficulties – specific conflicts among the functions and between the product and its intended market. Yet it is also a time when the

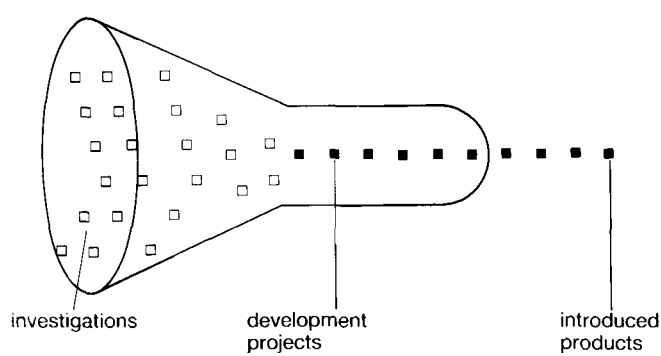
best management can hope for is to correct obvious problems, not pre-empt them.

The type of pre-project involvement needed on the part of management ranges from resolving policy issues, establishing functional and business strategies, and linking individual development projects to setting overall corporate and business goals. Ideally, the pre-project activities establish the context and boundaries within which each project is to be conducted and achieve its desired performance. Managers from several levels of the organization and from all of the functions have relevant inputs to make during this pre-project phase.

One of the reasons that managers do not spend more time in pre-project activities – establishing these boundaries and the context for individual development efforts – is that they have relatively few tools and approaches to do so. It is much easier to react to a physical prototype of the product or to specific problems, ranging from difficulties in pilot manufacturing to feedback from customers on sample products, than it is to anticipate how the development team should be guided and influenced while still allowing sufficient creativity and flexibility to arrive at a good, manufacturable design. Two concepts that can be easily applied in a variety of situations and provide a means for management to make inputs in this pre-project stage are the development funnel and functional maps[3].

The Development Funnel

The concept of the development funnel, as illustrated in Fig. 2, provides a framework for thinking about the generation and screening of alternative



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing.* New York: The Free Press, p. 295.

Fig. 2 The New Product–New Process Development Funnel

[3] *Dynamic Manufacturing*, by Robert H. Hayes, Steven C. Wheelwright, and Kim B. Clark (New York: The Free Press, 1988). See Chapter 10, "Laying the Foundation for Product and Process Development", pp. 273–303.

project ideas. A variety of product and process concepts enter the funnel for investigation, but only a fraction of those progress to the point of formal project approval and execution. Those that do must be examined carefully before entering the narrow neck of the funnel, where significant resources are expended on developing them into commercial products.

Managing the funnel involves two very different tasks. First, the organization must expand its knowledge base and its access to information in order to increase the number of new product concepts. This is analogous to expanding the size of the mouth of the funnel. Systematic thinking on the part of management as to how that can be done – from accessing research labs for more technical ideas, to purchasing concepts through licensing arrangements, to soliciting creative inputs from across the organization, its customers, and its suppliers – is one of the ways management can add value in this pre-project phase.

The second, complementary task, and one which often tends to dampen the creativity desired and needed so the firm will have the best set of options to choose from, is that of screening and selecting the handful of projects that will actually be pursued. This is analogous to narrowing the neck of the funnel while ensuring that a constant stream of good projects flows through it. Achieving an effective balance between creatively widening the funnel's mouth and rigorously narrowing its neck is not easy. Companies that do it best tend to combine various idea-generating mechanisms with a sequential review process.

A particularly useful step in applying the development funnel concept is its application as an investigative weapon. Several firms have gotten together a broad cross-section of managers involved and having influence on development projects, explained the concept, and then broken them up into small groups, asking them to diagram the funnel applied in their organization. Even a brief forty minute period doing so can provide tremendous insight as to the nature of their development funnel and its practical realities. Usually such graphic representations have several humorous aspects, yet provide accurate descriptions of the strengths and weaknesses of the firm's existing approach. In one scientific instruments company, where half a dozen groups of fifteen managers each developed such graphic representations of their firm's development funnel, it quickly became apparent that there were a handful of common characteristics that management needed to address:

1 While the firm obtained ideas from many sources, the idea generation effort was neither managed nor guided.
2 There was no defined start point for development projects. The company did not make systematic choices about which ideas became projects and obtained the resources needed for their eventual completion.
3 One of the sources of new product ideas was third-party suppliers, but their ideas didn't enter the funnel until just before market introduction. Thus they tended to get very incomplete development attention and review, resulting in a stream of problems (and even failures) after market introduction.

4 Within the supposedly narrow neck of the funnel, there were several bulges which reflected late redefinition of projects, fuzzy start dates for projects, the mixing of invention with application, changes in project team players, and the fact that early decisions often did not stick late in the procedure.
5 Many ideas that were killed partway through the funnel seemed to continually show up again at the mouth of the funnel as someone's pet project.
6 At the end of the funnel, top management added tremendous heat (and resources) to finally get individual projects completed.
7 Not many products came out the end of the funnel, particularly not many in a recently added product family. For neither the traditional nor the new family were projects being completed in a timely, paced manner.

A key point with regard to the development funnel is that every firm has one, but most firms do not examine it systematically nor do they consider how it might be altered to provide a clearer context and boundaries for each individual project.

In addition to developing an appropriate procedure that implements the desired "development funnel", management needs to relate individual projects and ideas to the ongoing business strategy and strategic choices within the functions. Mapping is a concept and technique for doing do.

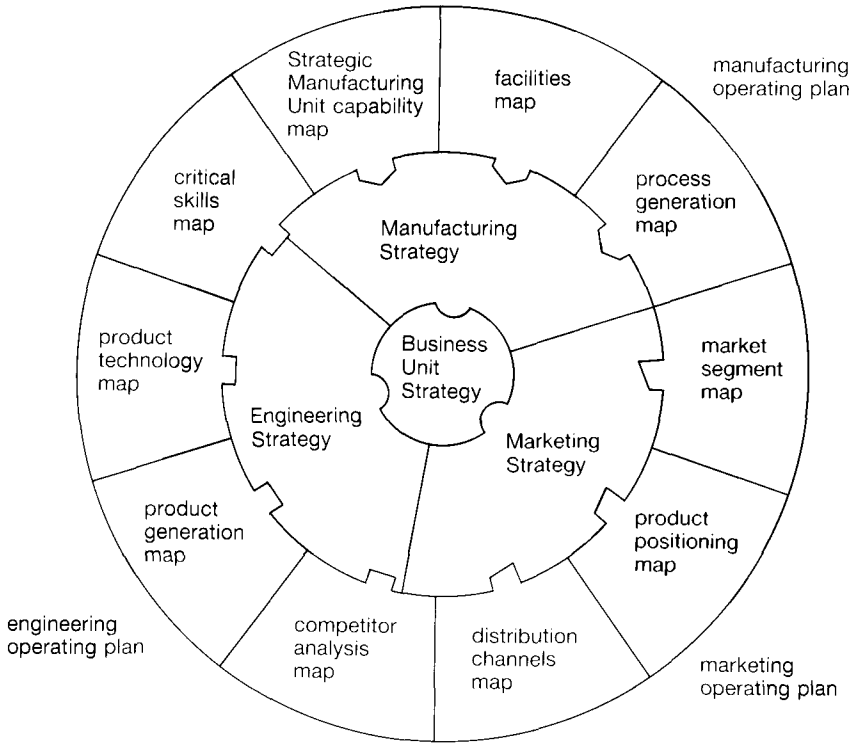
Functional Maps

A project of any significant size involves a number of people from various functional groups. For the project to achieve focus and move ahead rapidly, the entire project team must have a common or shared understanding of the objectives of the project and how it relates to their own functional area, the product line strategy, and the overall business strategy. While the topics of business planning and strategy development have received considerable attention over the past decade, most organizations concentrate attention on business strategy, usually as driven by the dominant function (for example, engineering in an electronics firm or marketing in a consumer packaged goods firm). The reality is that most groups in a firm do not have clear functional strategies.

Several of the "classic" problems encountered by development efforts and outlined earlier result because of gaps in planning between the business level and the functional level. With clear functional strategies, it is much easier for the project team to resolve policy issues and relate their efforts to the ongoing activities of the organization. The development of a set of functional maps can be an effective approach for developing functional strategies and, more importantly, relating them to the product development funnel and individual projects to be undertaken by the firm.

A functional map is simply a chart or a graph that depicts the behaviour of a critical variable (or combination of variables) over time either within the firm

or in relation to competitors. *Knowledge maps* summarize what is known historically about a particular variable or issue, and *strategic choice maps* suggest alternative directions that the business or function might pursue in the future (and often where the development project might play a critical role). Functional maps might include those illustrated in the outer ring of Fig. 3.



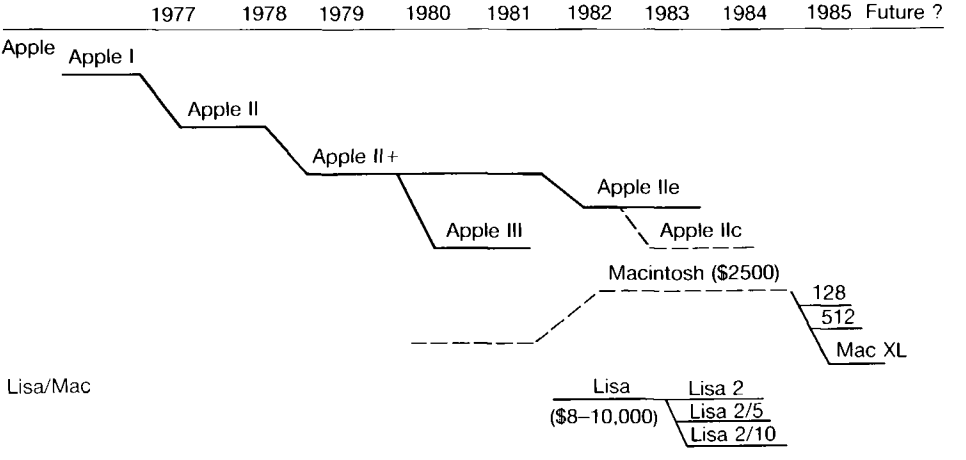
Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 284.

Fig. 3 Relationships between Business and Functional Strategies: Functional "Maps"

The complete set of functional maps indicate how various aspects of each function will interact and influence the project and its objectives and, in turn, how the project will help achieve strategic objectives. Thus manufacturing, with its concern for process technology, facilities, critical skills, sourcing, and plant location, might choose to develop maps on each of those dimensions. In the marketing function, issues related to market segmentation, product positioning, distribution channels, and pricing might each be areas where functional maps would be particularly relevant to planning development projects. Finally, within engineering, the application of new technology, the relationship of platform or core products to derivative products in the

product line, and the development of the proper mix of engineering skills and talents would all be candidates for functional maps.

Perhaps the single most useful map in establishing the context for a development effort is the "new product development map"[4]. An example of this map for Apple Computer, covering the period from their start as a business in the late 1970s up through the mid-1980s, is shown in Fig. 4. (Figure 4 is a *knowledge map* in that it reflects historical choices and evolution as opposed to future plans.) With the introduction of the Lisa and Macintosh family in 1984, Apple had two product families, each with multiple offerings.



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 293.

Fig. 4 Product Generations at Apple

As suggested in Fig. 4, the Apple II family may have been conceived originally as simply the next generation product for the low-end education market. In fact, it has proved to be extremely durable and while evolving, has continued to serve as the platform product for that family of offerings. The Apple III was an attempt to develop a related product for the business market, but was dropped not long after its introduction because of quality problems. The product map for the Lisa/Macintosh line indicates that Lisa was envisioned originally as the platform product for the family with Macintosh as the derivative product. However, Lisa proved not to be a good match with the market place, and Macintosh became the surviving product in that family[5].

[4]See "The New Product Development Map", op. cit.
[5]A more complete set of maps for Apple Computer is provided in Chapter 10 of *Dynamic Manufacturing*.

With regard to future product offerings at Apple Computer, in the late 1980s the firm announced two important decisions which internally provided the basis for developing a *strategic choice map* of future product generations. The first was that the Apple II would continue as the primary product for elementary and secondary education. The second was that an engineering work station version of the Macintosh (designated the Macintosh IIx) would become the platform product for the Macintosh family. Thus, future Macintosh development efforts would include a next-generation work station and then a set of derivative products aimed at the business and university markets. The company also has indicated that peripheral products (such as laser writers, file servers, and other hardware) will form a third family of product offerings. (Apple has subdivided its development engineering efforts into three groups, each focusing on one of these product categories.)

IV Project Management

An effective pre-project stage with appropriate managerial inputs and links to functional strategy can do much to raise the probabilities of a successful development project. However, there still remain a number of issues and tasks that must be dealt with effectively during the project management stage. While most companies do spend the bulk of their development resources during this stage, many do so late in the project when it becomes apparent it is falling short of original objectives. As a result, firms often forget the good thoughts they had about how to manage the project, and "Get it finished, never mind how" becomes the driving force. As a consequence, much of the conventional wisdom regarding effective project management is ineffective and, in many cases, simply wrong. Three very generic – yet powerful – aspects of project management can be used to illustrate both the challenge and the opportunity for effective management during this phase. These include overlapping problem solving, conflict resolution, and project organization[6].

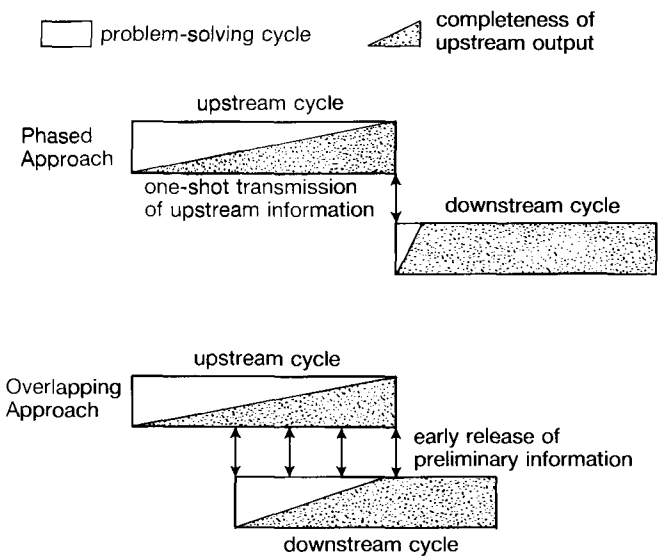
Overlapping Problem Solving

At a number of critical junctures, new product development projects involve the interaction of both an upstream and a downstream group. Early on the upstream group is often marketing, who specifies desired product characteristics for design engineering, the downstream group, who must convert those into technical solutions and designs. Subsequently, design engineering

[6] Many of the ideas in this section build on the work of Professor Kim Clark and his study of the worldwide automotive industry. That study, involving approximately 30 new car development projects in 20 different firms, has been reported in a number of Harvard Business School working papers by Professor Clark and is currently being converted into book form. I am grateful to Professor Clark for his contributions to my own thinking in this area. (See also Chapter 11, "Managing Product and Process Development Projects", pp. 304–339 in *Dynamic Manufacturing*.)

becomes an upstream group, and manufacturing a downstream group. At this interface, design engineering must transfer designs, or at least preliminary thinking about designs, to manufacturing, who converts those into manufacturing processes and procurement arrangements.

Even within any single function such as design engineering there may be subfunctions that have an upstream and a downstream relationship. At each such interface, there are issues involving problem solving and information transfer that must be addressed. As suggested in Fig. 5, at the extremes of a continuum, there are two very different modes for carrying out this interface – a *one-shot, batch approach*, and an *overlapping, intensive, two-way communication approach*. Each of these deserves further explanation.



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 314.

Fig. 5 Linking Problem-Solving Cycles

In the one-shot, batch approach without overlapping, the upstream group works through the details of their assignment and tasks and then – and only then – passes all of that information (in a single batch) to the downstream group. At this point the downstream group start to prepare and carry out their set of tasks. Commonly, this is referred to as the "over the wall" connection between the upstream and downstream groups. Not only does the upstream group avoid tipping its hand before they have completed their tasks, but the downstream group avoids making any commitments or doing any preliminary work until they have those final plans in hand. In many cases, this approach can lead to a number of repeat cycles as the downstream group identifies the need for modification and revisions in the work of the upstream group.

In the overlapping approach, with intensive two-way communication, there is a continuous transfer of information (and, more importantly, partial information) from the upstream to the downstream group with feedback and comments flowing in the reverse direction. In the case of the design engineering/manufacturing interface, the designers share with manufacturing their thinking as they carry out the design process, and manufacturing, in turn, provides feedback on that thinking and seeks to add value to the final design. Obviously this approach is the most effective one when carried out appropriately.

Preliminary study suggests that the overlapping approach can involve as much as five times the level of information transfer as the batch process. Thus, it must be perceived as adding value – to both parties – or else it will not be sustained. Manufacturing, as the downstream group in this example, must develop an ability to "forecast" the path design engineering is following, based on early clues, and then provide feedback as to the feasibility of that path. In addition, the downstream group, manufacturing must be flexible and adaptable, and must provide short cycle feedback on problems. It is not acceptable for manufacturing to take three weeks to get back to engineering on a preliminary idea. Nor is it acceptable for manufacturing to say repeatedly, "That won't work." Rather, manufacturing must become sufficiently skilled in the development procedures used by designers to be able to add value to their designs.

The extent to which the overlapping approach, with intensive information transfer, differs from conventional wisdom and common practice is perhaps best illustrated by how organizations using each of these two approaches characterize "Early Manufacturing Involvement" (EMI). EMI comes to the forefront rapidly as organizations explore ways to shorten new product development and avoid costly errors necessitating additional iterations at the eleventh hour. The following summarizes the characterizations typically given by those using a phased, batch approach (the traditional view) and those using an overlapping, intensive approach (the integrated view). It is interesting that within a firm using either approach, there tends to be great consistency in these views across both design and manufacturing.

Descriptions of "Early Manufacturing Involvement"

TRADITIONAL VIEW (Phased approach with batch information processing)	INTEGRATED VIEW (Overlapping approach with intensive information processing)
Manufacturing constrains engineering	Manufacturing enhances engineering
Engineering passes completed designs	Engineering and manufacturing exchange information continuously
Each function has separate responsibilities	Responsibility is shared throughout
Parochial functional attitudes	Team attitudes

Another illustration of the substantial differences in these two problem solving approaches comes from the procurement of parts in connection with a new product development effort. Conventional wisdom is that common parts across product families will speed up design. In an industry like automobiles this requires that suppliers be given a completed set of drawings and full specifications for the components they are to provide. This is a form of the phased, batch approach. Traditionally, even when unique parts were sought, the design was done by the automotive firm and the phased, batch approach was used to tell the supplier what to make and how.

Research by Professor Kim Clark suggests that the degree to which these two approaches have been followed varies dramatically among auto firms in Europe, the U.S., and Japan. In Japan, automotive firms source over half of their purchased components for a new car program from suppliers doing their own design of unique parts. While the automotive firm provides functional guidance as to performance requirements, the supplier is expected to know best how to design and build that component. This enables the supplier to do their own overlapping problem solving within their firm rather than be forced into the phased, batch approach. The resulting components are tailored to that new car program, and they can be more easily and more effectively produced by that supplier over time. In the U.S., less than 20% of the components for a new car program are sourced in this way. Over 80% are sourced in the more traditional way: the auto firm provides exact component specifications to the supplier and tells them how to make it.

Conflict resolution

An issue that ties closely to the problem solving approach used in a new product development project is the organization's approach to conflict resolution. The very nature of a development effort makes conflict inevitable. However, individual areas of conflict can be either healthy or unhealthy. The unhealthy kind arise when unilateral action is taken (usually on the part of the dominant function or strongest individual), legitimate discussion and questioning is suppressed, a group pretend they have a solution (often in an attempt to avoid conflict) before that is actually the case, or political considerations dictate the solution adopted. Healthy conflicts arise when knowledgeable people have different information, perspectives, or analysis, or when tough choices are required. The resolution of healthy conflicts, whether between functions or within a single function, are what the process of convergence to a final design is all about. Such conflicts need to be addressed in a systematic and effective manner.

Many organizations have discovered that five rules provide excellent guidance in resolving conflicts during the course of new product development projects. Recognizing these and having the discipline to adhere to them makes a significant difference to overall success.

1 *Conflicts should be surfaced as early as possible.* There are always more options early on, and the likelihood that the best solutions will be adopted is directly related to when conflicts surface.
2 *Conflicts should be resolved through mutual accommodation.* Political clout and functional dominance are not likely to lead to the best solution in most cases. Rather, thorough and reasoned discussion is needed.
3 *Conflicts should be resolved at the lowest level possible.* That is usually where the best information exists, where mutual accommodation is more likely to occur, and where conflicts can be surfaced early. When conflicts are pushed up the organizational hierarchy, two things tend to happen and both are bad – it takes longer, and the resolution tends to become more political and less based on fact.
4 *The lowest level of the organization must be made competent.* This requires skill development, training, and experience in the depth dimensions as well as the breadth dimensions of product development.
5 *Adhere to and follow through on agreed to solutions.* Nothing undermines the conflict resolution process more than false or pretended resolution.

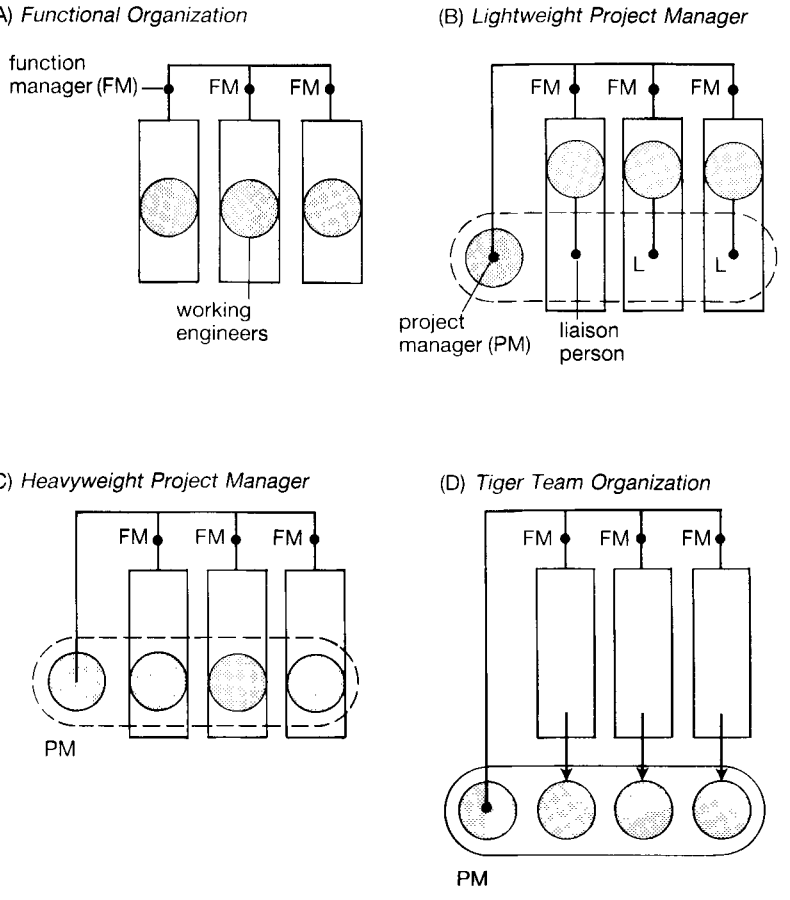
Project Organization

Effective development project management requires that the organizational groups involved develop the specialized capabilities needed *and* that the efforts of those groups be appropriately integrated. How a firm chooses to organize a development project can have significant bearing on both. As shown in Fig. 6, there are four basic ways to organize development efforts, each having its distinctive strengths and weaknesses[7].

The traditional *functional system* is depicted in the upper left. This approach, the most common in use today, organizes people together by discipline, each working under the direction of a senior functional manager. For instance, within each engineering discipline, specific engineers specialize in various aspects of the product under development. The work of the different functional areas is then coordinated either through a set of detailed specifications agreed to by all at the start of the project, or by occasional meetings where issues that cut across groups are discussed. Primary responsibility for the project passes sequentially – although often not smoothly – from one function to the next. This approach appears to work best where deep functional expertise is critical to the success of the project and/or where the firm has such a commanding position in the market place that they can dictate, to competitors and customers alike, the timing of new product introductions.

The second approach shown in Fig. 6 can be referred to as the *lightweight project manager system.* In this approach, people physically reside in their traditional functional settings, but each functional organization chooses a

[7]See Chapter 11, "Managing Product and Process Development Projects", in *Dynamic Manufacturing*, pp. 311–323.



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 320.

Fig. 6 Types of Organizations for Development Projects

person to represent it on the project team. These liaison representatives work with a lightweight project manager (lightweight in terms of status, command of resources, and experience) who has responsibility for coordinating the activities of the different functions and keeping track of the project schedule. Experience suggests that this approach is not particularly stable and is very dependent on the personality of the project manager. However, it is the one most commonly adopted when the functional system begins to falter in effectiveness.

The third organizational model is that of the *heavyweight project manager system.* In contrast to the lightweight system, a heavyweight project manager has direct access to and responsibility for the work of all the major players

on the project. This person is a heavyweight in terms of both status and level in the organization and in terms of control over the resources applied to the development effort. While the long-term career development of people on the project may still reside with the traditional function managers, the project manager takes on a very important role in their intermediate term evaluation. The heavyweight system tends to be robust and effective, but suffers in that most firms do not have career paths that develop such project managers nor do they have an abundance of people already qualified. In addition, if a firm has operated under the functional system for many years, the heads of the various functions may not initially be willing to give up power and influence to a heavyweight project manager.

The fourth form of project organization is the *tiger team system*, shown in the lower right of Fig. 6. Under this structure, the different functional areas assign and dedicate their people to the project team. These people are co-located throughout the project. The project leader is really a general manager, but rather than being charged to work within the existing organization in negotiating resources and achieving compatibility across product generations and different projects, the tiger team project manager is largely autonomous. Thus the tiger team tends to work best when the project involves a radical new product or a new market, and where issues of compatibility and transition to the operating organization are not major concerns.

There are many other dimensions of effective project management, but these three – problem solving approaches, conflict resolution methods, and project organization structures – are among the most important. However, even when handled effectively on a given project, there remains the challenge of transferring experience and learning to other projects. This suggests a third stage of product development requiring management attention, one that appears to be more challenging even than the first two.

V Organizational Learning

The individuals involved in new product development projects always conclude that they have learned a tremendous amount during that effort. Even when the project has not gone well, they have a long list of things that they've learned. Unfortunately, very few organizations seek to capture or tap into such learning over time. Just as individuals need to grow and develop by learning from their mistakes as well as their successes, organizations must do likewise.

Managers offer many explanations as to why their organizations fail to learn from past projects – the urgent need to reassign key resources to the next project (often before the current one is finished); the separation (physically, organizationally, and psychologically) of different functional groups, which inhibits cross-functional learning; the natural resistance to change, in any

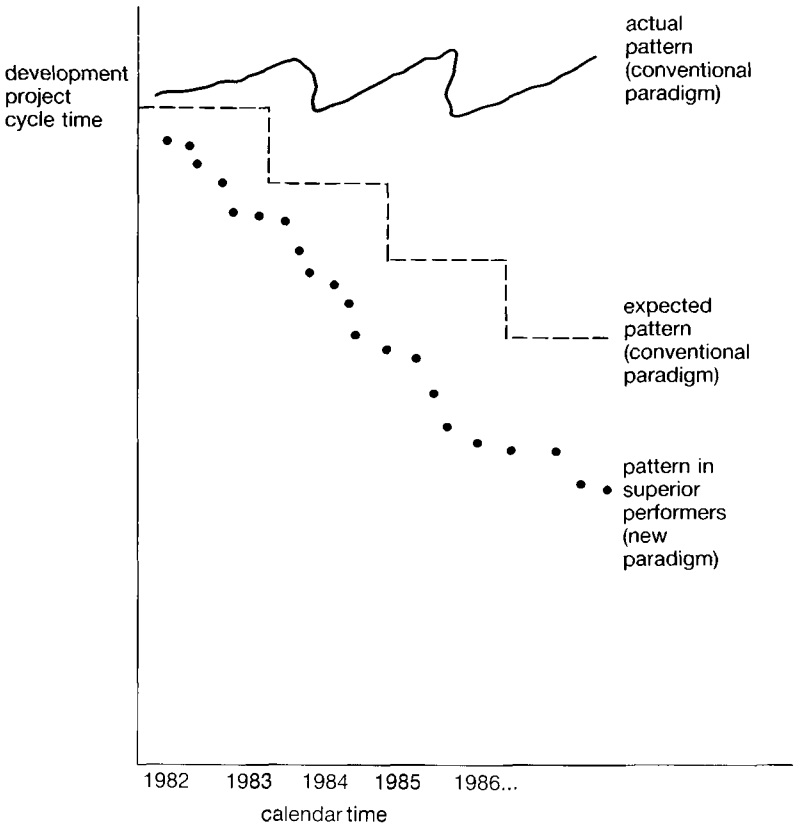
organization; and staff and system support groups' preference for fine-tuning the status quo. Furthermore, as one very successful CEO of a high tech company warned, "You'll never get anywhere looking across a series of development projects because no two are the same. It's all personalities, and even when you get someone good, you cannot be sure they will do well on the next project. There are just too many variables and too much random noise to make sense of anything."

While there is obviously some truth in each of these objections, they are much more an indication of why organizational learning is so difficult, yet so important, than an indication that it is impossible. Rather than make it less valuable, these challenges strengthen the competitive advantage achieved by those firms who can master the ability to "learn across product development projects".

One reason firms do not learn well across development efforts is because of the perspective they take regarding such learning and improvement. As illustrated in Fig. 7, the conventional view is that to the extent such learning occurs, it is a staircase process. Basically, a set of procedures is followed over the course of several projects, the lessons from experience are collected, and then that set of procedures is revised with the expectation that there will be a measurable drop in the time required for product development (and a measurable increase in the effectiveness of the product development effort).

Unfortunately, recent studies suggest that firms who operate with this perspective face a quite different reality. Instead of staying on a horizontal path until management finds a way to improve project management, performance usually deteriorates (cycle time drifts upwards). This is caused by the way managers in these organizations respond to problems in each individual project. As unexpected difficulties arise or problems are encountered (for example, someone forgets to double check something, or does not get a necessary sign-off), managers add steps to the existing procedures to make sure that those particular problems do not recur. As a result, procedures become more and more cumbersome, and project performance deteriorates. Eventually the organization decides that its procedures have become too bureaucratic and need to be streamlined. A major effort to improve those procedures is made, but generally it only brings the organization back to where it was before that cycle started. The sawtooth horizontal pattern shown in Fig. 7 generally fits reality much better than the expected staircase improvement pattern.

The relatively few organizations that do learn effectively across product development projects appear to seek smaller but more frequent improvements, as indicated by the dotted line in Fig. 7. They manage to learn from each project they undertake, continually streamlining and integrating the overall set of procedures. Even more important, they are constantly building and reinforcing the capabilities needed for future improvements.



Source: Hayes, Robert H., Wheelwright, Steven C., and Clark, Kim B. (1988). *Dynamic Manufacturing*. New York: The Free Press, p. 337.

Fig. 7 Two Approaches to Development Improvement

One of the areas in which these superior performers seem particularly effective in capturing individual project learning as an organization is in prototyping. Most new product development projects go through several prototype iterations as the product passes from concept to breadboard and on to prototype and pilot production unit. The superior firms do fewer cycles of prototyping, yet learn more in each cycle. It is not a matter of simply dropping prototyping cycles – which exist because the firm needs to learn more before moving to the next iteration. Rather, it is a matter of learning more from a given cycle so that fewer total cycles will be required.

These same firms also have discovered ways to shorten the feedback loops in prototype cycles so that each cycle occurs more rapidly. A combination of fewer cycles, each of which is faster, can do wonders for the overall product development cycle time. Equally important, shorter feedback loops and the

focus on maximizing learning in each prototype iteration reduces project costs, most of which are fixed period costs – thus the shorter the period, the lower the cost – and improve the resulting product performance because learning cycles are more complete, focused, and effective. Those organizations that are significantly faster at product development than the bulk of their competitors got there not by cutting corners but by being smarter, and the results show up in all aspects of their development project performance.

One of the ways the most successful firms tie together all of the concepts described in this paper is through the articulation and refinement of a *product development strategy*. Such a strategy consists of several elements, as outlined in Fig. 8. As part of the pre-project phase, the firm addresses, in a systematic manner, issues of technological forecasting and competitive assessment, linking and summarizing those in the form of specific goals. These serve to relate product development in important ways both to the firm's technology strategy and its business and product line strategies. As suggested by the right hand side of Figure 8, these firms also learn across projects, thereby continually improving their management and execution of individual projects.

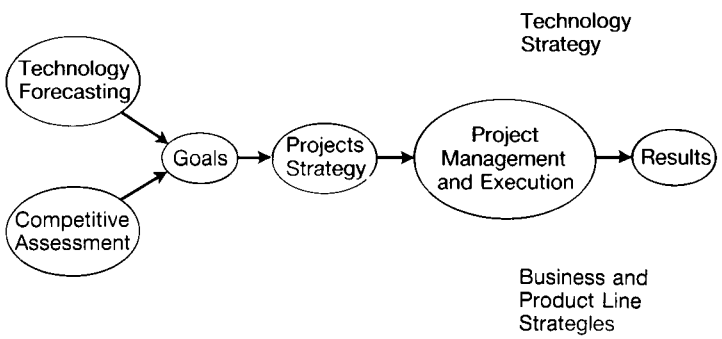


Fig. 8 Elements of a Development Strategy

Clearly a central element of all of this is developing what can be termed a projects strategy. Unfortunately, very few firms seem to have focused much time or attention on this. One way of thinking about a projects strategy is that it outlines for the organization the relationship between individual projects and the other elements of an overall development strategy. To do so effectively, it needs to include four very important subelements of a plan to continuously improve the firm's product development capabilities. These include:

(a) Outlining the path by which the fundamental capabilities needed for new product development will be provided in the future.
(b) A projects plan that indicates the sequence of projects and their goals, and how that sequence will operationalize and accomplish both the technology strategy and the business/product line strategy.

(c) Outlining the path and providing the means by which appropriate sources of knowledge will be accessed and developed across a series of projects.
(d) Relating the projects themselves to the learning and improvement path the organization wants to follow in building its new product development capabilities over time.

In one company where at least an initial effort has been made to develop a projects strategy, it was done by looking at a broad range of very specific goals across three subsequent product generations. Those goals ranged from the number of component parts in each generation of product to the yield at final assembly, and from the resulting service call rates to the operating cost structures for each product generation. Knowing that a new product generation would be required every two years for the next decade, management could plan how specific goals would be achieved through each generation and how, over time, the long range objectives would be accomplished.

Honda North America is another firm where the elements of a development strategy have been effectively developed and applied. They have become a leader in their global industry at reducing the design-production cycle. They can now:

- launch a new model in 18–24 months, against their competitors 3–8 years, while increasing quality, not merely maintaining it;
- launch complex engine modifications in less than 8 months time;
- implement the results of new product development efforts quickly, such as switching the entire supply chain over from the current model to the next generation model in a single weekend.

Not only have Honda accomplished this in North America, but they have achieved similar results in Europe, South America, and the Far East. As a firm, they have taken to heart the challenges of time-based competition and new product development. They have developed a capability that results in significantly faster and more effective new product development efforts than the majority of their competitors. Furthermore, in the process, they have instilled within their own organization the desire to continually improve and enhance that capability and use it in other, non-automotive, product market areas.

Time to Market in Manufacturing

David Saxl
ICL Manufacturing & Logistics, Kidsgrove, Staffordshire

Introduction

Following the plenary session, delegates to ICL's University Research Council Workshop on *Time to Market* formed themselves into five syndicate groups to discuss different aspects of the overall subject. Each syndicate group had a number of objectives, including the identification of opportunities for research or collaboration arising from the discussion.

Four of the five syndicate groups concentrated on the phases of

- Marketing
- Design
- In-plant
- Logistics

and the fifth group concentrated on the subject of organisation which is relevant to all of the other phases. This paper was produced following the discussion in the organisation syndicate and can therefore be taken as representative of all of the syndicate groups. The key points emerging from the other four syndicate groups are presented in the form of an Appendix to this paper.

Time to Market and competition

An observer, looking back at manufacturing over the period of this century, would probably have a view that competition for the first 50 years was essentially based on product innovation and engineering quality. This may not actually be the case, but the effect of looking back in time is to lose sight of everything apart from the actual design of the product being examined and one therefore assumes that this was the basis of competition. Looking at the second half of the century however, there has been an increase in the power of management theory, and the basis on which companies have been competing with each other has certainly changed. Emphasis has moved from portfolio management to price competition, then quality competition until now, in 1989, competition is increasingly being seen as being based on time.

If a company engages in time based competition this will have a number of effects on the style of operation within the organisation. Some of these effects will be discussed later.

Time based competitiveness may also affect the external view of the organisation both in terms of customer service (faster response to orders etc.) and innovation (first into the market with new product ideas etc.). The carrier firm Federal Express can be seen as an example of "service", having used speed of delivery to derive competitive advantage in their market. Honda can certainly be seen as an example of "innovation", using speed of new product introduction to defeat first Yamaha and now the European Auto Industry. The American company Atlas Door can be seen as an example of the synthesis of both of these approaches, using its ability to design and introduce products quickly to provide the customer with a solution exactly meeting his requirements and in competitive timescales.

This design and product introduction cycle is often known as "Time to Market" and it is this which is expected to be the principal area of competition for many companies in the electronics and computing sector.

Components of "Time to Market"

One of the difficulties of discussing Time to Market is defining an unambiguous start point for the cycle. This will be the point at which someone in the organisation perceives a market opportunity which is subsequently translated into a product. Unfortunately, in most companies this does not correspond with a formally recorded or measured event. The process can usually be broken down into a number of phases which are conducted more or less serially. These are:

- **Marketing**
  Starting from the bright idea, producing a business case, getting it approved by the organisation, and turning it into an agreed statement of marketing requirement.
- **Design**
  Starting from the statement of marketing requirement and producing an agreed design which meets the requirements of marketing and manufacturing.
- **Manufacturing**
  Taking the agreed design and turning it into a product which can be, and is being, manufactured.
- **Logistics**
  Setting up a physical distribution channel for the product; its associated support requirements, for instance literature, spares and service.
- **Market introduction**
  Ensuring that the market is ready to receive the product and the sales force is ready and equipped to sell it.

The recent focus on time to market as an area of competitive differentiation has produced evidence of large differences between the norm and best practice in various areas of manufacturing. Research in the automobile industry, for example, has shown a typical model introduction time of 43 months in Japan against a European and American average of 63 months – see the reference to Professor Kim Clark's work in Professor Wheelwright's accompanying paper. A difference of 2 to 1 has been observed in some cases in this industry, with even greater differences in the electronics industry. A frequently quoted example is Compaq's performance in launching the Deskpro 386 personal computer only six weeks after Intel announced the 386 chip.

A number of factors have already been identified as contributing to the differences in time to market. These factors include use of tools, types of organisations, culture of organisations, size of organisations, etc.

Types of organisation

Most large enterprises are organised on a functional basis. The logic for this form of organisation is that, by grouping all similar specialists together, it maximises the use of staff resources, allows areas of special competence to be built up and creates units which are large enough to allow for investment in tools and training.

An alternative form of organisation is product or market focused. Such an organisation brings together all the skills and resources needed to deal with a particular product or customer into a single unit which, in theory at least, looks just like a small single product company. (See Fig. 1.)

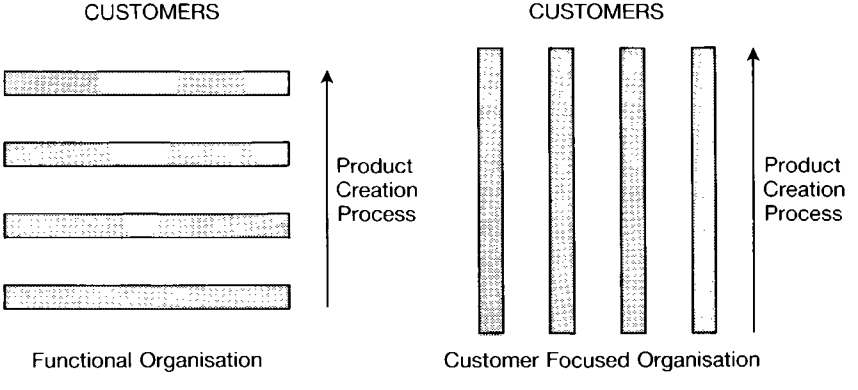


Fig. 1.

The use of projects within large organisations is an attempt to combine the best features of functional and product organisations. The existence of project teams can therefore be seen as either the perfect general solution to

organisational issues or evidence of the failure of processes which should make project teams unnecessary.

Any functionally based organisation must have clearly defined processes to succeed in product creation. These processes ensure that the right things are done to the product at each stage. They also ensure that it is passed from one function to the next as it is being developed, or as the service is being delivered. If similar organisations compete with each other they will do so on the basis of the quality of these processes and the size of the functions. The processes will generally determine the quality and timeliness of the product being created. The size of the functions will give competitive advantage to the largest organisation because of economies of scale.

A product or customer focused organisation has less need of formal processes to ensure that the product is created or the service delivered. Everyone in the organisation is focused on this single objective with no conflicts caused by having to deal with other products or other customers. Similar companies of this type will essentially be competing on the quality of their people because the importance of processes is reduced and the advantage of scale eliminated. In other words, with this form of organisation, the large enterprise has no inherent competitive advantage.

It has generally been observed that when product developments are carried out in functional type organisations, more than 60% of the development time is spent at the interfaces between functions. Many theories have been advanced to explain why functionally organised enterprises are slow at time to market and why they lose so much time at functional boundaries.

One factor is thought to be the complexity which is generated by the attempt to handle, simultaneously, activities for different products, for different end customers and from different suppliers. Since all of these factors are variables, they combine together to generate great complexity. The effect of different, competing project managers is to create an environment of continuous replanning in which more time may be spent adjusting priorities and queues than performing useful work.

Another factor is thought to be the rigidity of the functional organisation. It may offer efficient redeployment of resources but it is not able easily to provide resources to other organisations or to accept unplanned work. This is because the objectives and funding of the group are often so tightly defined, to maximise control of the process, that the manager dare not take on any activity which may threaten achievement of his objectives.

Given the limitations of functional organisations, it would be natural to assume that if the basis of competition was to be speed of development, then the enterprise should be organised in a completely product or customer orientated fashion in order to minimise the time lost in crossing functional boundaries. This suggests that small organisations will be much better than

large ones in respect of speed of new product development and this corresponds with observations made by many researchers in this field.

The implication of this for a large enterprise is that time based competition would force it into the kind of organisation where it loses all competitive advantage over its smaller rivals. It is this problem which needs to be addressed.

For a large enterprise to gain competitive advantage from scale while being organised in a product or market focused manner it must look for, and develop, opportunities for sharing and synergy between the market focused segments. One area could be in the provision of support services built into the infrastructure of the organisation in such a way that they are available to all product groups without presenting any organisational implications. In the same way as no-one questions the sharing of a telephone exchange in a current organisation then no-one should question the provision of a library service in a future one. This library must contain information about re-usable elements of a design. A current example is the use of symbol libraries within electronic assembly. Obviously there are already many other examples of libraries and the concept will be extended into other areas in the future.

It follows that such libraries, which previously would have been considered as "nice to have", must now be considered as key sources of competitive advantage. Libraries, and the processes to create, maintain and make them available, might well be considered proprietary in the future, rather than generic items to be provided at least cost, as they have been up to now. At the moment libraries tend to contain only generic items. In the computer industry this means commercially available components and commonly used software routines. In the future one would expect the library to stretch further up the value chain to contain information about complete sub-assemblies and large software modules. These library items would be re-usable by other projects, not because they were explicitly designed to be packages, but because the process used in their design resulted in a re-usable product whose attributes were stored in the corporate library.

Apart from provision of infrastructure services, another factor for success of product or customer focused organisations is the flexibility of the workforce and the nature of internal communication. This type of organisation involves frequent changes of group size and composition and there must be no barriers to prevent this. Examples of such barriers include: too many levels of management, inappropriate objectives and budgets, and cultural expectations of organisation stability. These problems have been considered by Drucker, who concludes that future organisations will be very flat, relying much more heavily on IT systems for internal communication and control rather than following existing hierarchical, objective driven models.

A third success factor is likely to be the level of internal communication between members of different teams. Teams are unlikely to contain all of the

skills they need to solve all of the problems. At any given time the expert in a particular field is likely to be in the wrong team. His expertise must be available to the person who needs it. This does not mean that the expert is a full time consultant but where he can solve problems for other people quickly and easily he should be able to do it. Similarly, it is the responsibility of everyone in the organisation to know who are the experts in every field and be able to approach them without going through formal processes. Just as the organisation must allow and encourage this networking to occur the ability to operate in a network manner is a key attribute of every member of the workforce.

Possible research

Having considered these ideas, the workshop syndicate group dealing with organisation felt that there was an opportunity for further research into the relationship between organisation structures and information flows.

The information flows can be characterised as product related (vertical) and skill or experience related (horizontal). Either the vertical or the horizontal flows will encounter organisational boundaries depending on the form of organisation (see Fig. 2). The research could examine the factors which determine the effect of the boundaries on the information flows. These factors could include information systems, networking skills, levels of management, objective-setting processes, etc.

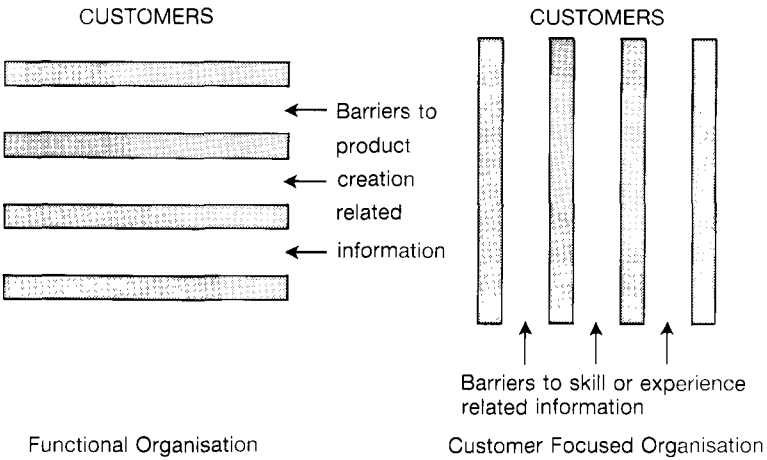


Fig. 2.

Summary

- Functional organisations appear to be slower than product or customer focused organisations for product development. Reasons may include the complexity of dealing with many projects for many customers and from many suppliers.

- Fast time to market therefore requires customer or product focused organisations rather than functional ones.
- Organisations of this type do not offer the large enterprise obvious economies of scale or competitive advantage.
- New sources of competitive advantage must be found for large enterprises. These could include libraries and infrastructure, information systems to support new forms of organisation, networking skills of individuals.
- Research opportunities exist to investigate the organisational factors affecting information flows.

Acknowledgements

The principal speaker at the University Research Council Workshop on "Time to Market in Manufacturing" was Professor Steven Wheelwright of Harvard University. The members of the "Organisation" syndicate were:

Tony Dimond	City University
Tim Gillespie	ICL
Professor Tom Lupton	Spain
John Panter	ICL
Chris Phoenix	ICL
David Saxl	ICL
Dr Ulbo de Sitter	Adviesgroep Koers
Professor Nigel Slack	Brunel University
Greg Wojtan	ICL

Appendix

The Marketing syndicate considered that there was considerable scope for computer systems to aid the marketing process. Additionally, a large benefit could be derived from having a clearly understood process model to remove some of the uncertainty from the marketing process and allow for a closer communication between marketing, design and manufacturing.

The Design syndicate considered that there was a large opportunity for computer systems to aid the design process. The other conclusions, however, were similar to those of the organisation syndicate, focusing on the need for generalist skills and on the need for people to be process rather than organisation focused.

The In-plant syndicate highlighted the need for appropriate performance indicators to measure the manufacturing plant contribution to time to market. Ideas for reducing time to market included concentrating on the transfer of manufacturing experience to the design process, either by transferring people or by creating design tools which contained manufacturing experience. Another opportunity for research or collaboration seemed to be the development of evolutionary software control systems, which would

allow the organisation to change and develop in line with changing requirements, rather than be constrained by existing IT solutions.

The Logistics syndicate identified a special problem with the provision of services which are very people-intensive. Even though it may be possible to compress the time to market cycle for products, the cycle for services will be extremely dependent upon people and the reprofiling of these people is almost always a lengthy process.

DATA SECURITY

The VME High Security Option

Tom Parker
ICL Defence Systems, Eskdale Road,
Winnersh, Wokingham, Berkshire, England

Abstract

Criteria for evaluating the security properties of computer systems are now well established and widely accepted. The security capabilities of ICL's VME Operating System have recently been enhanced in accordance with the requirements they lay down, by the addition of a High Security Option (the VME HSO). This paper describes the VME HSO concentrating on the features that have enabled it to achieve a high level of security certification by the British Government. The product is also aimed at the commercial market, and the paper describes the integrity, audit and usability features that have been provided to satisfy requirements in this area.

1 Introduction

ICL's VME Operating System has established itself over the years as being one of the more secure commercial operating systems available. A previous paper[1] in the Technical Journal described the ways in which the software and hardware architectures of VME and its host systems combine together to provide the fundamental structure upon which secure higher level functions can be built.

That paper was written seven years ago, and VME has since then moved forward a long way. At the time of the paper, ICL was making initial proposals to the Department of Industry (as it was then called) to develop with their financial support a purchasable set of security enhancements. The development was to be aimed at both the government and commercial user populations; it was to provide usable security, offering a great deal of flexibility in the choice of security policy; it was to provide strong security conforming to recognised independent security quality standards and it was to be evaluated by a recognised authority, resulting in a written certificate that ICL could use in the marketplace. In December 1984 ICL had the go-ahead to develop the product, and work started in 1986.

The development came to be known as the VME High Security Option, usually called the VME HSO. It is available on Series 39 machines.

2 Measuring How Good it is

On a large commercial computer system the amount of code to be executed in the performance of its duties is typically very large indeed. It is responsible for a wide variety of complex tasks aimed at helping the end user to do his job but at the same time constraining him to use the system only in authorised ways.

The operating system and the applications it is supporting act as both policeman and provider, and it is often very difficult to examine the whole of this complex mixture to make sure that the security it provides is acceptably robust.

In the past, various techniques have been applied in a somewhat ad-hoc manner to assess the quality of security provided by a system. Some examples are:

- straightforward functional testing of security features;
- examination of source code listings, sometimes aided by automatic tools;
- "tiger team" attacks in which security experts attempt to penetrate the system in a simulated real-life situation;
- consideration of architectural and design quality; a well designed and well structured system is both easier to assess and more likely to be correct;
- formal verification of the security properties of the system using mathematical techniques; such techniques are at present feasible only on small systems designed and written in special languages.

2.1 Standard Evaluation Criteria

Until the 1980's there was no way of obtaining a reliable standard measure from these techniques; there was no concept of marks out of ten or position on a scale of security. However in August 1983 the first official set of standard criteria for the evaluation of computer security was published by the US Department of Defense[2]. This standard is widely applied in US Government procurements of secure systems and its influence is now pervading European government and commercial requirements. Although the evaluation scale has been subjected to criticism relating to its scope of application and its too close coupling of functionality and correctness requirements, it is a remarkable technical achievement. It represents the culmination of a decade of research, and it is the only universally recognised scale available today.

The scale comprises the following set of levels, given in increasing order of security quality:

Level D No security.

Level C1 Basic security, suitable for the control of a relatively benign user population in low-risk environments.

Level C2 Strong conventional security. A discretionary security policy is in force, under which each user is responsible for defining the access controls that apply to the data he owns.

Level B1 Support of a mandatory centrally imposed security policy. Defence against corrupt application code is possible.

Level B2 Level B1 strengthened by a more rigorous and comprehensive approach and a better security structure.

Level B3 A system specially designed with security as its main overriding priority. The security critical code is specially constructed to make it as compact and easy to evaluate for correctness as possible.

Level A1 Similar to B3, but all of the security critical code in the system has been designed and built using a language to which mathematical techniques can be applied for formally verifying the security properties of the system.

A complex set of criteria apply at each level covering the system's design, functional capabilities, testing, development environment and documentation. The names of the levels are chosen to highlight fundamental differences between the different letter categories. To increase a system's security quality from one overall category to another is intended to be a major step forward (e.g. from D to C1, or C2 to B1).

An evaluation scale has also been developed by the British Government[3]. This has built on the experience obtained from the American work and is more flexible in its application. One important feature is its ability to separate out questions about what the system does, from questions about how well it does it. Other scales are emerging from the German Government[4] and from the Department of Trade and Industry in the UK[5].

Over a period of two years, starting in early 1987, the VME HSO has been subjected to an intensive security evaluation by an independent technical team of security experts funded and controlled by HMG. ICL's aim was to obtain certification for the system at a UK equivalent of level B1 on the American scale, with a similar but more complex rating on the HMG scale. This was successfully achieved in May 1989 for version SV221 of the system. ICL is now discussing ways in which this certification can be carried forward into subsequent versions.

At the time of writing this paper, ICL knows of no other proprietary general purpose operating system which currently offers this level of security functionality and assurance.

3 Level B Systems – Why So Much Better?

The single most significant advance in the transposition from a level C to a level B system is the support of a centrally controlled Mandatory Access Control Policy, which provides for data confidentiality without relying on the good behaviour of either end users or the application software they use.

Such a policy possesses three major features:

- it allows a security manager to determine and mark the levels of confidentiality of data held by the system;
- it allows the manager to determine and mark which users are cleared to access what data according to its confidentiality markings; only if a user is "cleared" to the level of confidentiality of the data is he permitted to read it;
- using a rule known as the "information flow" rule it prevents untrusted application code from circumventing the above checks by maliciously copying data from a highly confidential file to a less confidential file which a user with a low clearance can subsequently access. Such code (sometimes called "Trojan Horse" code) may attempt to do this unknown to the user for whom it is executing.

The first two features are relatively easy to provide. The standard approach is to associate a confidentiality label with each data object (its "Classification") and with each user (his "Clearance"). When a user requests to read an object his Clearance is compared with the object's Classification and access is permitted only if the former is higher than the latter†.

The third feature however, is at the same time the most significant and the most difficult. Its significance lies in the fact that without it, it is not sufficient to evaluate only *some* of the code in a system to be sure of the system's security; without it, *all* code needs to be evaluated. The latter is one of the major deficiencies of level C systems; the ability to distinguish between "trusted" and "untrusted" code is the big leap forward that level B systems make, both in terms of assurance and in terms of their basic evaluatability. It is only on level B systems that a user can execute some unknown code on his confidential file and be sure that its contents cannot be leaked by that code to any user not permitted to see it.

The protection offered by information flow control is difficult to achieve because its straightforward imposition can be too restrictive in real application environments. An operating system cannot be expected to understand the internal logic of all of the application code that might ever be executed on it, so it must lay down information flow rules in ways that do not depend in any way on such an understanding. These rules will necessarily be restrictive in nature; they will be a blunt instrument that, unless used with great care, might cause legitimate operations to fail because the system cannot be sure of their legitimacy.

An example may help illustrate this point (see figure 1). Suppose a particular application is performing two "update by copy" operations in parallel. The

†More precisely, this rule and the information flow rule are defined in terms of a "dominance" relationship between the security labels associated with the source and destination. The first formal statement of the MAC rules was derived by two American researchers, Bell and LaPadula[6].

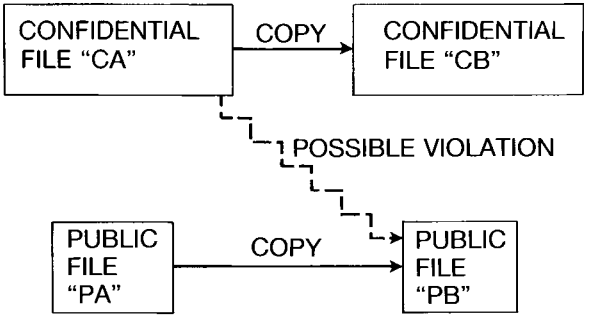


Fig. 1. Information Flow Example

first involves reading a confidential file CA and copying it with appropriate changes to a second confidential file CB; the second involves reading a publicly available file PA and selectively copying it to a similar public file PB. Neither of these operations constitutes an information flow violation, but the Mandatory Policy cannot permit the application simultaneously to open CA for read access and PB for write access *in case* it maliciously, or accidentally copies data from CA to PB. The operating system does not know that the application will not do this, and it certainly does not trust it not to do it. Indeed if the application had been tampered with and contained maliciously written Trojan Horse code, it would be very likely to do something of this kind.

So the problem is not that information flow security is difficult to achieve (at least at B1 levels of assurance), but rather that it is difficult to be sure that applications will still work when information flow controls are imposed. More significantly, the working of the system itself might depend on the continued functioning of standard system "application" processes which might suffer at the hands of the information flow rule.

The next Section describes how ICL solved these problems in VME.

4 Usable Flow Controls

Naturally, the VME HSO supports a Mandatory Confidentiality Policy with information flow controls. To do this it uses security labels along the lines already described. The system is designed so that a security manager can choose names for the labels that are appropriate for his needs, and they may be chosen to describe either or both hierarchic confidentiality levels and non-hierarchic confidentiality compartments. The basic flow control rules outlined above are applied, but enhanced in a number of ways which minimise their impact on the system's usability:

- it is possible for a suitably trusted user to mark code so that the Virtual Machine it is executing in is permitted potentially to violate the information flow rule. It is expected that installation management will themselves

ensure that no actual flow violations would occur. The legitimate application in the example could be marked in this way to enable it to work, but only after it had been suitably vetted to be trustworthy in this respect. For this reason the marking is said to belong to a group of markings known in VME as "trust" markings. VMs executing in possession of any of these trusts are known as "Trusted Processes".

- it is possible for a system administrator to run an application in a way which allows it to continue to work as if no Mandatory Policy controls were being applied, but which audits all cases where violations would have been caused had these controls been in force. In the example application the opening of PB for write access would trigger an audit message because CA had already been opened for read access.
  This information can be used in two ways: either the application can be reorganised to complete and close down its PA to PB copying before opening the confidential files – in which case not even a potential violation is caused and the application can now be run under full controls with no security problems, or the application can be marked as trusted as described above, with the exact reason for the need for trust having been identified and the verification of the code's trustworthiness having consequently been made easier. This "trial" mode method of running an application is intended to be used primarily as a transition aid.
- it is possible for a security manager to configure the Mandatory Policy for the system so that flow control is not policed at all. For many commercial systems who perceive that the threat of information comprise by this means is small this would be an appropriate choice.
- it is possible for a security manager to constrain information flow to be permitted potentially to occur only within a band of confidentiality levels. In the example it could be arranged to permit potential CONFIDENTIAL to PUBLIC information flow, but prevent any flow from, say, SECRET files.

5 Integrity

Although secure commercial organisations have a strong interest in protecting sensitive information from getting into the wrong hands, their major motivation for obtaining and installing a secure system is usually that of prevention of fraud. One of the technical consequences of this is that there is a need to provide data integrity, and it is therefore at least as important for a secure operating system to provide strong data integrity controls as it is to provide strong data sensitivity controls. A particularly significant paper on this topic was recently published by Clark & Wilson[7].

Because of this importance, and despite there being no specific requirement to do so in the American evaluation criteria, ICL has implemented a Mandatory Integrity Policy in the VME HSO, to complement its Mandatory Confidentiality Policy. The integrity features closely parallel the confidentiality features, and are as follows:

- the policy allows a security manager to determine and mark the level of integrity of data held by the system.
- it allows the manager to determine and mark which users are cleared to *modify* what data according to its integrity level. Only if a user's integrity clearance is higher than the integrity level of the data is he permitted to modify it.
- it prevents high integrity data from being corrupted by low integrity inputs; this rule is an integrity dual of the confidentiality information flow rule. It is Trojan Horse *data* rather than Trojan Horse code which is defended against here.

There is however an important difference between the two policies. The Mandatory Confidentiality Policy has no interest in distinguishing between different untrusted application code modules that may be used to access highly confidential data; there is no concept of giving code a clearance. The integrity policy does have this concept, and there is therefore a fourth feature:

- it allows the security manager to determine and mark which code modules are cleared to *modify* what data according to its integrity level.

By making use of this rule the security manager can be sure that important application data is operated upon only by the proper authorised application code under the control of a properly authorised user. Furthermore, the integrity flow rule prevents such an application being spoofed into running with unauthorised input data.

The VME HSO allows the security manager a high degree of flexibility in the way in which this policy is applied. The enforcement options are similar to but separate from those available for the Mandatory Confidentiality Policy. In particular the integrity information flow rule which prevents the flow of low integrity data into high integrity data can in real situations be relaxed if the code involved has been produced to defend itself against spoof input or other low integrity input, or if other features of the operating environment can be used to guard against supply of the wrong input data.

Finally a word about viruses: high integrity software on a VME HSO system is protected against modification by the same integrity marking as that given to that software when it is executing; in this way the Mandatory Integrity Policy ensures that no software of lower integrity could modify it. This means that a virus, which reproduces itself by copying itself from software module to software module will always be confined to at most the integrity level of the software module, within which it is introduced into the system. The operating system code of the VME HSO is protected by a special integrity label which customers do not use for their own data files. This code is therefore protected against corruption from any viruses that may be unknowingly introduced on customer program files. Similarly, a security conscious customer site, by ensuring that all unknown software is introduced only at a very low integrity level until it has been given a clean bill of health, can guard its own software against viral attack.

6 Other Mandatory Policy Refinements

There is a variety of other ways in which it is useful, and sometimes essential to allow the use of a Mandatory Confidentiality or Integrity Policy to be applied to fit a particular installation's needs.

Once the concept of being able to mark code, or nominate users, as trusted has been implemented, one can identify a further set of privileged functions that can be controlled using different trust markings. Examples of functions that can be specifically controlled under the trust system of the VME HSO are:

- the ability to change a confidentiality or integrity label,
- the ability to set and change security controls,
- the ability to set and change audit controls,
- the ability to change one's own password,
- the ability to introduce alien magnetic tapes into the system,
- the ability to override the discretionary access control system.

Only users given the appropriate trust, using software marked with the same trust, may perform any of the functions listed above. In total there are nearly thirty different categories of trust supported on the VME HSO, allowing a fine degree of tailoring to be applied. Services and Tasks can also optionally be controlled by trust; for example users can be confined to a particular Packaged MAC service when operating with a particular trust.

There are two points of particular interest in the examples above:

The first is the distinction that is possible between the audit and security management functions; this distinction allows a clear separation of responsibility to be enforced between these two important roles.

The second is the way in which trusts are used to strengthen the discretionary access controls of the system. Is is now no longer possible to penetrate the system by illegally obtaining access to the privileged SWITCH-USER or SET-PERMISSION-OVERRIDES commands. Such commands are now controlled under the Mandatory Policy. Only trusted users may use them, and then of course only subject to the privileged library constraints that have always applied. It is also worth noting that the power of these privileged commands is anyway significantly reduced in VME HSO systems; even if an untrusted user were to switch to become the Security Manager, he would be unable to exercise the powers of that user since it is only that user's discretionary capabilities that are acquired. The intruder would not have been given either the trusts or the clearances of the username to which he had switched.

The trust system has been extended in other ways. It is possible to nominate particular workstations as possessing or not possessing particular trusts. In this way users can be controlled in their choice of workstation from which they are permitted to perform their trusted actions. This idea is further

extended to encompass other communications devices like network gateways and cluster controllers, so it is possible for example to prohibit any trusted activities coming in from a gateway to a public X25 network.

The Mandatory Policy labelling scheme is also extended to cover communication devices and links, so it is possible to label a workstation with a particular confidentiality or integrity clearance which limits the effective clearance of users that use that workstation. For example, an installation could limit work on data with integrity category of PAYROLL to those workstations located in the payroll office.

Similarly, it is possible to mark devices like disc drives and line printers with security labels which constrain them to handling data whose security markings lie between defined boundaries. In this way, the printing of confidential or high integrity data can be confined to one or more nominated printers, and the storage of particularly sensitive files can be confined to nominated disc or tape drives. This latter form of protection can also be applied in terms of disc and tape partitions themselves, permitting fine grain control over file placement.

Finally, it is possible to mark Services and Tasks with labels so that whatever a user's clearance, for certain types of Service and Task it can be bounded according to the nature of the work being done.

7 Auditing

The ability to record for posterity what is happening on a system is almost as important as the ability to control it in the first place. Auditors wish to make system users accountable for their actions; they wish to analyse the ways in which a system is being used or abused; they wish to be able to look back to a record of previous events in order to assess damage done when a belated discovery of an attack on the system has been made; they wish to be able to monitor particular kinds of action or action by particular individuals or action using particular system access points in order to forestall attacks on the system; finally they wish to deter, to make sure that users know that their actions are being watched and recorded.

With these wishes in mind, ICL decided to transform the Audit capabilities of the system under the HSO. Existing audit facilities were supplemented by the provision of a complete new set of security audit records with special message types and subject to special protection.

The following events can be audited under the new scheme:

- attempted security violations,
- login, jobstart, session initiation and termination,
- submission of incorrect login data,
- changes to mandatory and discretionary security policy,

- changes to audit policy,
- the exercising of nominated types of trust under the mandatory policy,
- procurement of printed output,
- loading of code from nominated libraries,
- changes to passwords, but not the values involved,
- changes to security label values,
- any access by any user to any protected VME object.

The last of these event categories has the potential to generate an unacceptably large amount of audit data, so a wide variety of subsetting options has been made available to the audit manager.

These are:

- accesses by nominated users,
- accesses using nominated workstations,
- accesses using nominated services,
- accesses when code from a nominated library is available for execution,
- read accesses to objects whose confidentiality exceeds a nominated threshold,
- write accesses to objects whose integrity exceeds a nominated threshold,
- accesses to particular nominated objects or object types.

Together, these features permit an Audit Manager to define the precise audit profile he requires for his system. This can be varied on a day to day basis to react to changing circumstances.

In designing the formats of the new audit records ICL had a choice to make between human readable but large and therefore inefficient formats, and compacted machine processable formats that are more difficult for a human to interpret directly. ICL opted for the latter on the basis that in the future the raw data from audit trails will increasingly require extensive machine preprocessing in order to provide statistical data, to highlight significant events and to analyse for unusual changes in user work patterns that may indicate potential attacks on the system[8]. A demonstrator for analysing VME audit trails has already been developed by Logica in the UK under the auspices of the British Government's Central Computer and Communications Agency (CCTA).

8 Other Features

Space does not permit a full description of all of the security features of the VME HSO; indeed, it is ICL's policy to restrict the availability of full details of the product to bonafide customers having a legitimate interest in it, and for such customers a range of six security manuals has been produced. Other enhancements to the system's security have however been made in the areas of authentication, security labelling of printed output, and the protection of discarded data; this paper has only hinted at the full power of the "trusts" system.

A range of enforcement options has also been implemented to permit a customer to move gradually and painlessly into a secure mode of working following purchase and installation of the HSO.

9 What is the customer to make of all this complexity?

A question like this is understandable, and ICL has been very conscious that the power and potential complexity of the features provided by the HSO can be rather intimidating. The company has therefore provided a comprehensive training and consultancy support programme to supplement the HSO product itself. It should be remembered that it is very much in ICL's interest to make sure that the introduction of the leap forward in security that this product represents is a success.

It is also very important to note that a customer's philosophy when implementing his particular security policy should be "keep it simple and stupid!". Although the VME HSO provides a rich and complex variety of controls, they are there as a shopping list from which each different customer is free to select his own simple profile; the complexity is ICL's, not the customer's. If a system uses only a small proportion of the features offered by the VME HSO, but as a result, sensitive and valuable information is protected to a high level of assurance, then use of the product has been worthwhile. A customer should not feel that in order to get value for money, all of the security features of the system must be exercised to their fullest extent, indeed such an approach would be likely to achieve just the opposite effect.

10 What the VME HSO does not do

The VME HSO is an operating system development. It protects objects that are understood by the basic VME operating system. This means it protects things like whole files, libraries, disc partitions, communications devices and magnetic tape drives. It does not directly protect application-level objects like individual data items in an IDMS or Ingres database. To VME a database is a file, or at most a few files. The database is protected therefore at exactly that level of granularity and its individual data items are protected by the HSO only as a consequence of the protection afforded to the files that contain them.

Similarly, any software that runs on top of VME is treated by the VME HSO as being untrusted unless it is explicitly told otherwise by the security manager. This normally means all application packages and all superstructure products including TPMS, are treated by the VME HSO as untrusted. This is not to say of course that these products are not worthy of any trust; indeed many installations will utilise their protection features to supplement the security provided by the VME HSO, whose features might then be looked upon as providing the secure environment within which individual applications can operate.

11 Next Steps

At the time of writing of this paper only a few customers have obtained experience of this new product. Choice of future enhancements will therefore depend very much on feedback which is yet to be obtained. There are however two areas of future development which are worth highlighting:

The first is in the area of FTAM, or "File Transfer and Access Method". An early implementation of this feature on VME will permit VME HSO customers to transmit a file's security label along with the file, and therefore allow such a file to be protected in its destination system in the same way as on its source system.

The second development is in the area of user authentication. In the distributed systems of the future, users will wish to authenticate themselves once to the system as a whole, and use the results of this to access all of the applications they wish, no matter which particular end systems contain them. ICL is developing such a Network Authentication Server and the VME HSO will be adapted to accept the resulting certified identities rather than repeat the authentication process by engaging in VME login exchanges. VME usernames will then become resources which individual users may or may not be permitted to access. By this means accountability of individual human users will be achieved no matter how many share the use of the same VME username.

12 Conclusions

The American evaluation scale and its UK equivalent represent a major step forward in our understanding of what is required of a secure computer system. Level B systems on these scales will give a significantly better level of security protection than the conventional level C systems of today. These benefits will not come without effort on behalf of both users in managing their systems securely, and manufacturers in giving them the technical tools with which to do this; the potential gain in our ability to protect data in computer systems is however enormous.

Of prime importance is the support of a Mandatory Confidentiality Policy, without which the higher level of security assurance provided by level B systems cannot be obtained. The commercial world requires similar assurances with respect to integrity, and it is possible to satisfy these requirements by providing support for a Mandatory Integrity Policy. In both cases however, the implementation needs to be rich and flexible; a simple implementation would have unacceptable usability consequences.

Access control is not by itself sufficient; users must be made accountable for their actions and an audit capability of similar power and flexibility is also required.

The VME High Security Option provides these and other related security facilities for ICL's customers. It has passed its first hurdle: successfully

achieving British Government evaluation to the UK equivalent of B1 on the US DoD scale. The next hurdles may be even more difficult: proving its usability, manageability and security against real attack in real customer environments.

References

1 PARKER, T.A.: Security in a large general-purpose operating system: ICL's approach in VME/2900, ICL Tech. J., 3(1), May 1982.

2 DOD 5200.28-STD.: Trusted Computer Systems Evaluation Criteria, Fort George Meade MD USA: National Computer Security Center, December 1985.

3 CESG: UK Systems Confidence Levels, CESG Computer Security Memorandum No. 3, Issue 1.1, Feb. 1989.

4 GERMAN CIPHERBOARD: National Catalog of Criteria for the Evaluation of Trusted IT Systems, Draft version, issued by the German Cipherboard, Federal Republic of Germany.

5 DTI: Evaluation and Certification Manual, V23 – Version3.0, one of five volumes produced by the DTI Commercial Computer Security Centre, RSRE Malvern, Worcs.

6 BELL, D.E. and LAPADULA, L.J.: Secure Computer Systems: Unified Exposition and Multics Interpretation, MTR-2997 Rev.1 MITRE Corp., Bedford, Mass. March 1976.

7 CLARK, D.D. and WILSON, D.R.: A Comparison of Commercial and Military Computer Security Policies, in Proc. Symp. on Security and Privacy, IEEE, April 1987.

8 TERESA F. LUNT: Automated Audit Trail Analysis and Intrusion Detection: A Survey, in Proc. 11th National Computer Security Conference, Baltimore, October 1988.

Security aspects of the Fundamental Association Model

Heather Alexander
STC Technology Ltd, Newcastle-under-Lyme
David McVitie
International Computers Ltd, West Gorton, Manchester

Abstract

This paper reports some results from the application of a formal specification and prototyping method called **me too** [1, 2] to aspects of federated computer systems, based on a discussion document produced by ICL's Systems Architecture Group [3]. It has been written in the light of experience with **me too** specifications and prototypes of the concepts in that paper, since the development of these more formal specifications forced clarification of several key issues. This report does not contain the actual specifications developed during this collaboration, as it is essentially a rewritten version of [3]. Details of the specifications and how they influenced the contents of this report can be found in [4].

1 Definition of concepts

1.1 Servers and environments

A federated system consists, amongst other things, of a number of *Servers* which, as the name suggests, offer some kind of service to the overall system. Servers operate within *Environments*, each of which may hold some data concerning the operation of its servers and other servers in the system. For example, part of the local data may describe routes between servers in the various environments.

1.2 Initiators, Responders and Connecters

Every connection between two servers is asymmetric in that the server at one end starts the connection process. This server is called the *Initiator* while that at the other end is called the *Responder*. The term *Connecter* is used as a general way of referring to either of these servers.

The Fundamental Association Model describes how two servers (a Responder and an Initiator) become associated in the system. The Responder decides to offer a service of some kind to the federated system and creates an

instance of that service (a Service Instance, or SI). It sends details about the service availability to other servers in the system in a package of information called a token. To create an association between the Responder and another server, that server (the Initiator) must receive the token, note its details and accept it into its pool of tokens. Communication between the associated servers is only possible when a connection is established across the association. Only one connection may exist across the association at a time, but there may be many such connections during the lifetime of the association. One aspect to be considered when creating a connection is security, the subject of this paper.

1.3 Security aspects

Security is described by means of *Security Levels*, which may be possessed by servers or by routes between servers (at this level of the model). However, it is important to remember that the systems under consideration are *federated* systems; that is, they are distributed systems with no central controller or data store. Consequently, there is no single repository of wisdom about security levels in the system. In order to assess the security level of an entity in the system, we have to consider the various locally-held views of it. A local view of a level is called a *Rating*; that is, a rating is a concrete indication of how a server or route is seen – it has a particular value which can be set, inspected and/or manipulated.

A rating is made up of *Security Factors*, where each factor can have one or more values associated with it. Thus, a factor might be Colour with the values red, blue, yellow, or Day with the single value Monday. A factor is expressed as a set of such values (in the usual mathematical sense of an unordered collection of unique elements). Thus, these factors can be written as Colour = {red, blue, yellow} and Day = {Monday}. A rating is, in turn, considered to be a set of factors associated with their names, so that one server might have a rating of

{Colour = {red, blue}, Day = {Monday}}

while a second has

{Colour = {red}, Day = {Sunday, Monday, ..., Saturday}}

Intuitively, we would expect this to mean that these two servers could communicate for activities rated

{Colour = {red}, Day = {Monday}}

but not for ratings which include the factors

Colour = {blue}

or

Day = {Tuesday}

These informal ideas for combining ratings and deciding on allowable connections will be made more precise in sections below, but first some terminology is required.

1.4 Exact and inexact factors and levels

A factor that contains a single value is called an *Exact* factor; a factor with more than one value is *Inexact*. A level which is described by a rating containing only exact factors is an exact level; any other level is inexact. An inexact level may contain both exact factors and inexact factors.

1.5 Inclusion of ratings

If A and B are security ratings, then A *includes* B if each factor in B is present in A, and if the values of each factor in B form a subset of the values for the same factor in A.

1.6 Default ratings

There are some special cases to consider, each involving some omitted information:

1 a connecter's environment has no rating for some entity involved in the connection process,
2 a rating omits one or more of the possible factors,
3 a rating contains a factor with no associated values.

There are two possible approaches to dealing with omissions: prohibitive and permissive. That is, omission can mean "I cannot (or will not) talk to anyone about this security rating or factor" or it can mean "I do not know about this security rating or factor and, further, I do not care about it". The approach taken here is generally permissive, with the result that an omitted rating or factor is deemed to have been defaulted.

The rationale behind this treatment of defaulted factors is based on the view that security is protected by the one who knows what is required and can therefore take the responsibility for that security. If a factor is not known to an object, its security rating cannot contribute to any discussion of that factor, and so the factor is omitted. Thus, in building the composite picture of a security level, those who know about a factor contribute that knowledge to the level from their ratings. Where a factor is omitted by a requester of a service (the Initiator), the success or failure of the request will depend on the evaluation of that Initiator by other objects involved in supplying the service. For example, suppose the Initiator omits the Colour factor in trying to talk

to a service supplier (the Responder) whose own rating contains that factor with values {blue, green}. This will only succeed if the responder rates the Initiator to include the factor with values {blue} or {green} or {blue, green} or if it is defaulted.

Taking this view, the special cases listed above are handled as follows:

1 if a rating is omitted, this is equivalent to it being present with all factors defaulted,
2 if a factor is omitted from a rating, this is equivalent to that factor being present with all possible values; i.e. an omitted factor implies that the factor does not constrain the security aspects of the connection,
3 a present but empty factor is not allowed. Prohibition can be explicitly requested by the special factor value "NULL", which says the object thus rated will not talk to anyone who tries to mention that factor. An alternative method is for individual ratings to include a special value which they interpret as prohibitive values.

1.7 Trusted & untrusted connecters

A *Trusted* connecter is one that is able to discriminate between security levels. This ability is indicated by the fact that its rating contains at least one inexact factor. An *Untrusted* connecter is unable to discriminate security levels in any way. It is constrained to operate at a single point in the multi-dimensional space of possible security values; that point is defined by the exact factors making up its rating. An alternative way of expressing this is that an untrusted connecter *must* operate at an exact level.

Note that a connecter may be trusted with respect to some security factors and not others – since some degree of trust is involved, it will still be referred to as a trusted connecter. The environment containing a server may limit the trust invested in it.

1.8 Quality of Service (QOS) parameters

A connection request may involve the specification of a *Required Security Level* for the connection. This level may be exact or inexact. If an inexact level is supplied, successful connection will result in the return of a *Security Level Achieved* in order to inform the requester about any restrictions imposed.

It is assumed that parameter values are not expressed in terms of physical or logical properties, but that levels are matched onto routes by ratings whose values have taken these physical and logical properties into account.

1.9 Security ratings involved in connection

The security level of a server or a route is not known in an absolute manner by a single authority. Rather, as a connection is made, various views (ie.

ratings) of the connecters and the possible routes are combined in order to determine whether or not the connection may be made and at what security level it may conduct its conversations.

Ratings are held as part of an environment, and may relate to any servers in the system and to any routes from its servers to other servers. The following list indicates various ratings which may influence the security level achieved:

R1 the Initiator's environment's rating of the Initiator
R2 the Initiator's environment's rating of the Responder
R3 the Initiator's environment's rating(s) of the route(s) available to connect to the Responder
– there will be one R3 rating for each available route
R4 the Responder's environment's rating of the responder
R5 the Responder's environment's rating of the Initiator
R6 the Responder's environment's rating of the route via which the Initiator has connected.

Earlier versions of this list included local ratings held in the connecters. Since a connecter is only as secure as the environment which is protecting it, this general concept of local ratings has been removed.

1.9 Binding ratings together

The ratings for routes and connecters listed above have to be combined in some way during a connection, a process called *binding*. In accordance with the policy described in section 1.5, binding is permissive rather than prohibitive.

The general rule is that, for factors in common, their common values are retained in the result. The effect of binding therefore is to maintain or reduce the trustedness in a security level, since values can only be kept or removed. If the binding results in an empty factor, indicating that common factors had no values in common, then the binding fails. Note that this rule applies equally well to both exact and inexact factors.

Given that an omitted factor means it is present with all values of that factor, factors which appear in one or other of the ratings look as if they are being added to the result.

Some of the issues can be illustrated with an example. If **A** is the rating

{Day = {sat, sun, mon}, Colour = {red, blue}}

and **B** is the rating

{Day = {mon, tues}}

the result of binding them will be

{Day = {mon}, Colour = {red, blue}}

In this result, Colour has become explicit (where it was defaulted in **B**). Moreover, the combined rating has reduced the trustedness involved in **A** and **B**, since Day is now an exact factor and Colour is restricted to two values.

If one of the ratings is omitted, recall that this is treated as a rating with all factors present and defaulted, so binding it with another rating does not change that second rating at all.

2. Constructing a connection

This section describes how a connection takes place, showing how the various ratings are combined, a route is chosen and the Security Level Achieved is obtained.

First, though, consider the meaning behind all the activity involved in connection. The overall purpose is to create a connection capable of handling conversations at a particular security level, which may be exact or inexact, across an existing association between an Initiator and a Responder. In that connection, it is the responsibility of each end to protect its own security. Connections will fail if either end believes its security to be compromised.

When the Initiator is requested to set up a connection, the request will indicate the quality of the service required by means of "Quality of Service" (QOS) parameters. One of the QOS parameters which may be specified in the connection request is a particular security level (in this version of the model, this is the only QOS parameter considered). The information in the QOS parameter has to be combined with the Initiator's environment's views on the security levels at which the Initiator and Responder may operate (R1 & R2 in its environment). Binding QOS with R1 and R2 yields a composite view of the level required which will have the same or less trust placed in it as the QOS parameter requested.

The Initiator's environment then has to select a route capable of carrying conversation elements at that security level. If there is no such route, the required security level may be further reduced in order to establish a connection across one of the routes available. The result of this stage is a proposed security level for the connection which may be transmitted to the Responder end as part of the connection request.

The Responder and its environment also have views on the entities involved in the connection, and these are combined with the transmitted level in order to arrive at an overall agreed security level for the connection. If a connection is established successfully, this result can be returned to the Initiator so that it too knows at which level the connection is operating.

Diagrammatically, this process can be seen as shown below. Abbreviations refer to details of the steps, which are explained in the sections below. The numbered elements indicate the order in which the activities occur.

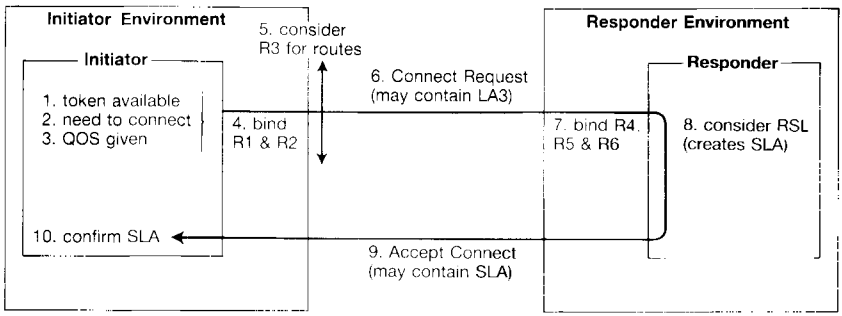


Fig. 1.

This diagram is explained in more detail in the following sections. Sections 2.1–2.4 and 2.8 describe the work undertaken at the Initiator end of the connection, while sections 2.5–2.7 describe the work undertaken at the Responder end.

2.1 Initiator sets the Required Security Level (RSL)

This is simply the QOS parameter, if it has been supplied.

2.2 Binding in the Initiator's environment (prior to choice of route)

The Initiator's environment binds the RSL with R1 and R2. If the RSL was defaulted, it will have no effect on the result of the binding; that is, it will not constrain the connection process at all. The result of this process is called *LA2* (Security Level Achieved at stage 2).

R2, the view of the Responder, is crucial to the success or failure of the connection, since it is effectively the Initiator's environment's view of the entire security discourse. In order to ensure that security is upheld, R2 should contain all the factors about which the Initiator's environment is concerned. If this is the case, LA2 will contain all the factors relevant to the connection with the Responder, be they exact or inexact, regardless of whether or not they were defaulted in other ratings at this end of the connection.

2.3 Choosing the route

The Initiator's environment now has to choose a route, based on the R3 ratings it has for possible routes. There will be a rating ($R3_i$), possibly defaulted, for each route i between the servers which are to be connected.

If a route i exists which can handle LA2 as it stands (that is, $R3_i$ includes LA2), then that route is selected. Otherwise, the route ratings are each bound in turn with LA2 to produce a series of LA3 ratings.

It is possible that there will still be an element of choice left after either of these steps have been taken, ie. if there are several routes capable of handling LA2 or several suitable $LA3_i$ are derived. In this case, a weighting scheme of some kind can be devised to act as a tie-breaker, but the details of such schemes are not relevant to this paper, since it suffices to note that a choice must somehow be made.

Note, too, that in choosing a route, the user of the Initiator may be penalised for "greed", ie. asking for more than is needed. This penalty can arise because the routes capable of handling the unnecessary levels may be more costly to use. Alternatively, the connection may be refused if the Responder cannot operate at the excessive levels requested or if there are no suitable routes to the Responder.

Another issue concerns whether or not a route can be enhanced in some way, that is, its security level can be changed by the use of some scheme, such as encryption. Enhancements may be enabled in order to obtain a preferred subset or may be necessary to make the connection possible at all. The mapping of enhancement schemes onto existing ratings is undertaken by the environment. This is not an explicit part of the current design, since, for the purposes of modelling at this level of detail, a simpler view is adequate. In this simplification, it is deemed that, to deal with the possibility of enhancement, R3 includes explicit ratings for enhanced routes. For example, if a route i can be enhanced in two ways, then it is considered that this produces *four* possible routes: i alone, i with enhancement 1, i with enhancement 2, and i with both enhancements. These four "routes", with their individual ratings, can then be treated in the normal way.

At the end of this route-selection stage, the value LA3 (possibly identical to LA2) has been determined.

2.4 Requesting the connection

In the next stage, the Initiator's environment requests the connection with the Responder. In the process, it has to judge whether or not the Responder can be trusted to receive information or make decisions about security levels. This judgement depends on its view of the Responder, as held in R2. If discussion is possible, security information from LA3 will be transmitted to the Responder as part of the connection request.

Where the servers attempt any discussion of security, they must have an appropriate overlap of concept, sharing the same security vocabulary. That is, for any factor transmitted to the Responder, the Initiator and Responder must agree on (at least a common subset of) the possible values of that factor.

At a minimum, this ensures that LA3 is understood at the Responder end and that the Initiator understands any level returned to it.

If R2 is exact, the Initiator's environment does not believe the Responder to be capable of discussing security, and so does not transmit any security information to it. In this situation, all the Responder's environment can do is ensure that its locally-held ratings (R4, R5 and R6) allow the connection to be made. This implies that the connection works or fails on the basis of local knowledge at each end without any interchange of security information. There is no need, therefore, for the two ends to synchronise their security vocabulary (but they do need to consider the security implications of their ratings, since there is an implication of equality between the ends).

If R2 is inexact, the Initiator's environment thinks that the Responder end is capable of discussing security levels and so it will transmit some or all of the security information contained in LA3 to the Responder as its contribution to that discussion. What is transmitted depends is based on the factors in R2. The inexact factors in the R2 rating represent the Initiator's environment's view of what the Responder has to comprehend about the required security level. Consequently, all the factors of LA3 which correspond to inexact factors in R2 are transmitted to the Responder end. Factors in LA3 which correspond to exact factors in R2 are not transmitted, since R2 implies that the Responder is not capable of discussing them. They can only impede successful connection, and, as for an exact R2, it would be necessary to ensure a common vocabulary across both connecters.

The net result is that the Initiator's environment only sends that security information from LA3 with which it believes the Responder can be trusted. If it does not trust the Responder at all, it sends no security information to it.

2.5 Binding in the Responder's environment

The Responder's environment binds LA3 (if sent), R4, R5 and the appropriate R6. The route has already been chosen by the Initiator, including any enhancement, and R6 is the Responder's rating of that choice. If the route has been enhanced, the R6 rating used will take that into account. The eventual result of this binding is *LA5* (Security Level Achieved at stage 5).

2.6 Responder accounts for internal rating

The Responder may have been given a rating for a particular Service Instance when it was created. This is the only local rating considered in the model, corresponding to the QOS parameter at the Initiator end. Once LA5 has been calculated, it is bound with this internal rating for the SI, resulting in LA6 (Security Level Achieved at stage 6).

2.7 Informing Responder and Initiator of Security Level Achieved (SLA)

If the connection has not failed yet, it will succeed. The Responder was the last object to handle the calculations of security levels so it is aware of the level achieved. If the Initiator discussed security with the Responder, then it has to be informed about what was achieved. The Responder knows when this is the case, because it is indicated by the presence of security information in the connection request. Thus, if some or all of LA3 was transmitted to the Responder, then LA6 is transmitted back to the Initiator.

2.8 Back at the Initiator end

If the Initiator's QOS parameter was inexact, then its environment must inform the Initiator about the SLA. This is LA6 if returned from the Responder, or LA3 if the Responder was not consulted. Of course, if the connection failed, no level is achieved.

3 Example

In order to highlight any misconceptions, this section describes an example. Consider documents to be sent across an email association, where the documents carry security codes of red, blue or green. A connection capable of carrying such documents from the email Initiator is requested, so that the QOS parameter is Code = {red, blue, green}.

The Initiator's environment agrees that the Initiator is cleared for these codes; that is, R1 contains the factor Code = {red, blue, green}. However, it considers that the Responder currently in use in the email association is only cleared for blue and green documents; that is, R2 contains the factor Code = {blue, green}. Accordingly, the Initiator reduces the security level, so that LA2 contains Code = {blue, green}.

For simplicity, assume that there is a single route available, capable of handling all three codings. In this case, the rating is unchanged by the choice of route and so LA3 is the same as LA2. Because the Initiator's environment believes that the Responder can be trusted to discuss these colour codes, this factor from LA3 is transmitted to the Responder as part of the target security level for the connection.

Assume that the Responder end imposes no further constraints and that the level LA6 achieved thus contains Code = {blue, green}. This is sent back to the Initiator which deduces from the security level achieved that it is not free to send red-coded documents across the connection.

This example has taken a simple case to begin with. Now consider some of the alternatives which could occur:

- suppose that the Initiator ratings are as above, ie. it trusts the Responder to know about blue- and green-coded documents. However, the Respon-

der knows it can only handle green codes (R4 contains Code = {green}). The connection will succeed, though only allowing the Initiator to mail green-coded documents, because the Responders knows the Initiator was entitled to talk to it about such codes. If R4 had contained Code = {yellow}, the connection would have failed, due to the binding rules.

- suppose the Initiator thinks (R2) that the Responder can only handle green-coded documents, but all the other ratings are as in the original case above. No security information will be sent in the connection request, so that the Responder does not know at what level the Initiator is operating. However, the local binding (of R4, R5 and R6) succeeds and so, therefore, does the connection request. No SLA is returned.
- suppose now that R2 contains Code = {green} and that R4 & R5 both contain Code = {pink}. This will succeed because all the ratings are exact. No security information is transmitted to the Responder so it does not know (or care) that the actual values are different.
- suppose that all the ratings at the Initiator end default the Code factor (including the QOS parameter), so that it is asking to be able to use the email association for any documents, regardless of colour coding. Since the Initiator does not know that the Responder cares about this factor (ie. it does not appear in R2), then the factor will not appear in the transmitted request. Since the Responder rating (R4) does contain the Code factor, the success of the connection depends on its rating of the Initiator (R5), which must also default the factor or successfully bind it with the values in R4.

4 Summary

This paper is seen as a step towards a detailed description of the security aspects of the Fundamental Association Model, and is a contribution to the high-level design of the federated architecture.

The description here is based on a formal specification in **me too**. In several areas, it was the precision of the **me too** notation and the experience gained from the prototype which led to this description. One example was the identification of the special case where disagreement over trust between connecters should not fail the connection. Although **me too** does not figure explicitly in this description of the model, it was essential to its clarification, illustrating one potential use for formal methods in an existing development culture.

References

1 HENDERSON, P., (1986) "Functional programming, formal specification and rapid prototyping" *IEEE Transactions in Software Engineering* SE-12, 2, 241-250.

2 HENDERSON, P., MINKOWITZ, C. J., (1986) "The **me too** method of software design" *ICL Technical Journal* 5, 64 95.

3 MCVITIE, D. G. (1987) "Quality of Service parameters for security control over connections" Discussion paper, ICL West Gorton.

4 ALEXANDER H., JONES, V. M. (1989) — forthcoming *Software Design and Prototyping using* **me too**. Prentice-Hall International, London, UK.

An Introduction to Public Key Systems and Digital Signatures

Jim Press
ICL Network Systems, Jays Close, Basingstoke, Hampshire, RG22 4BY*

Abstract

The increasing spread of distributed systems has led to a growing realisation of the need for security. Public key cryptography is increasingly being used in modern computer security techniques and international security standards. Yet, many people do not understand what they are and how they can be used.

This paper is intended as a simple introduction to the basic concepts of public key systems and some of their applications. Particular attention will be paid to the concept of digital signatures.

1. Introduction

Traditionally, the use of computer security techniques has largely been confined to the military and banking communities. With the advent and widespread adoption of 'Open Systems' standards and media publicity about the exploits of hackers and the threat of viruses, commercial companies are gradually realising the value of the information in their systems and the need to secure it from adverse influences.

Losses of information through fraud or malicious damage annually cost some companies thousands of pounds and even have forced some into bankruptcy, yet under 1 per cent of current UK networks have applied suitable security mechanisms.

The security of a distributed system must be a total solution and must address not only external threats to the system but also internal threats: up to 75 per cent of computer related crime is commited by 'insiders'.

Security must encompass the physical security of the system, administrative controls (roles, responsibilities, audit logs, etc), logical controls (who can access what and when) and data protection (privacy, authentication, integrity).

---

*Now working for ICL Cross Range Software and Services, Six Hills House, London Road, Stevenage, Hertfordshire, SG1 1 YB.

2. Security in open systems

The OSI Security Architecture [ref. 1] defines a number of optional security related services to provide for the protection of information within an Open Systems environment.

(a) *Authentication Services*
These provide authentication of communicating entities within an OSI layer.
- Peer Entity Authentication: to verify, at the time of usage, the identities of one or more communicating peers;
- Data Origin Authentication: to verify the source of received information;

(b) *Access Control Service*
This provides protection against unauthorised use of resources accessible via OSI.

(c) *Data Confidentiality Services*
These protect data from unauthorised disclosure.

- Connection Confidentiality: to protect all user data on a connection;
- Connectionless Confidentiality: to protect connectionless user data;
- Selected Field Confidentiality: to protect selected fields within connection-oriented or connectionless user data;
- Traffic Flow Confidentiality: to prevent any information being gained by monitoring traffic flows;

(d) *Data Integrity Services*
These protect data against active threats such as modification, insertion, deletion or replay of data.

- Connection Integrity with/without recovery: to protect all user data on a connection with or without attempted recovery;
- Connectionless Integrity: to protect connectionless user data;
- Selective Field Integrity: to protect selected fields within connection-oriented or connectionless user data;

(e) *Non-Repudiation Services*
Two kinds of non-repudiation service have been identified:

- Non-Repudiation with Proof of Origin: to provide the recipient with proof of the origin of data which will protect against the sender falsely denying the sending of the data;
- Non-Repudiation with Proof of Delivery: to provide the sender of data with proof of its delivery which will protect against the recipient falsely denying receipt of the data;

With the exception of the access control service, the mechanisms used to implement these services are based upon data cryptography. This will be illustrated in the following sections.

3. Conventional cryptography

In a conventional or symmetric cryptosystem, the sender encrypts a message (called the 'plaintext') by a transformation (the 'cipher') influenced by a secret parameter called the 'key'. The result of the encryption is the 'ciphertext' which is meaningless to anyone who does not know the value of the key.

The receiver of the ciphertext decrypts it by reversing the transformation using the same key as the sender, to retrieve the original message.

As long as only the intended recipient shares the key with the sender, the privacy of the communication is ensured (Data Confidentiality).

This process is illustrated in figure 1.

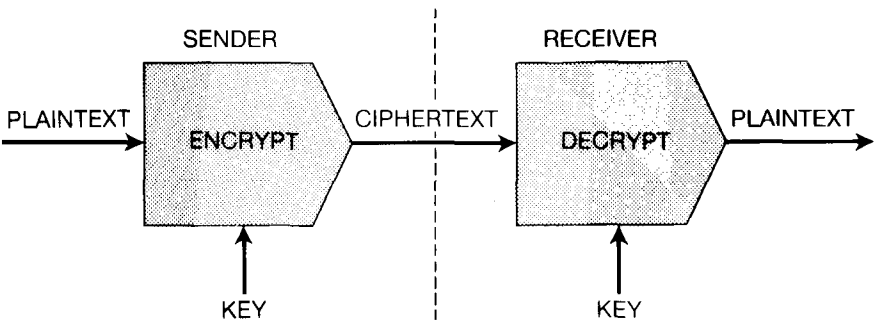


Figure 1 Conventional (Symmetric) Cryptography

The best known symmetric cryptosystem is DES, the Data Encryption Standard of the US National Institute of Standards and Technology (NIST, formerly the National Bureau of Standards), published by ANSI as X3.92 Data Encryption Algorithm. Other examples of symmetric cryptosystems include the ICL proprietary cryptosystems IPACRYPT 100 and IPACRYPT 200.

4. Public key cryptography

In a symmetric cryptosystem, both the sender and receiver of a message must share the same key which must be kept secret from anyone else. This gives the problem of how to distribute the key in a secure manner.

In 1976, Whitfield Diffie and Martin Hellman [ref. 2] considered the problems of key distribution for symmetric cryptosystems and came up with a method of encrypting data so that there is no need to distribute secret keys. Their ideas formed the basic concepts for Public Key Systems, also known as Asymmetric Cryptosystems.

In a Public Key System, each user (whether this be a human, an application or an entity in an OSI layer) generates two keys from a random seed using trapdoor one-way functions. A Trapdoor One-Way function is a function which is easy to compute but very difficult to reverse unless a certain piece of information (the 'trapdoor') is known; for example looking up someone's name in a telephone directory to find the number is easy, but looking up the number to find the name is hard. However, if you are told the first letter of the person's surname, then the problem is made a lot easier.

Since it is infeasible to compute one key from the other without the 'trapdoor' information which will be kept secret or even destroyed, there is no danger in making one of the keys public knowledge. This 'public' key can then be used by anyone wishing to send encrypted data to the user, who can then use the other 'secret' key to decrypt the messages. Since only the intended recipient knows the decryption key, privacy is ensured (Data Confidentiality).

Figure 2 illustrates the process of public key cryptography, where the trapdoor one-way functions used in the generation of the keys is denoted by 'OWF'.

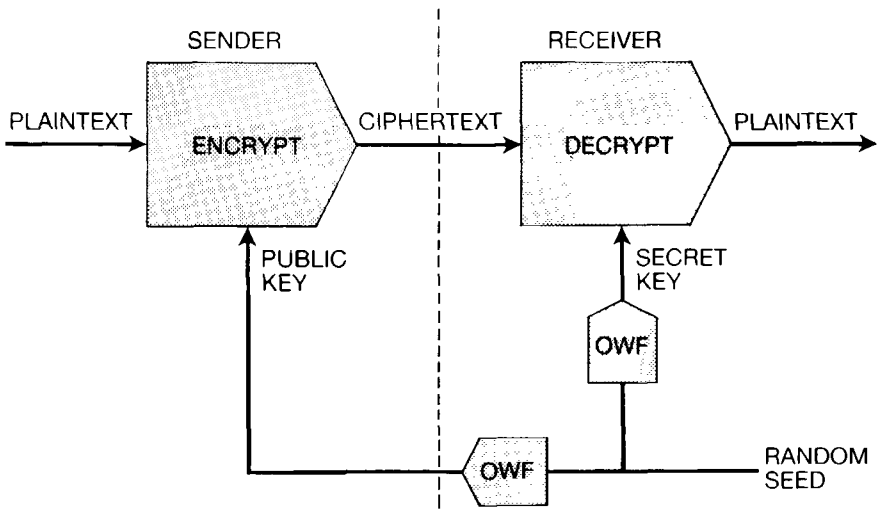


Figure 2 Public Key (Asymmetric) Cryptography

The most popular Public Key System was invented by Rivest, Shamir and Adleman, more commonly known as RSA [ref. 3]. A detailed description of RSA is given in the appendix.

Owing to the complexity of the mathematics, Public Key Systems are generally much slower than conventional (symmetric) cryptosystems and are therefore unlikely to replace them for providing data encryption in high

speed communication networks. For example, the fastest RSA chips can operate at around 64 K bits/second, whilst a modern symmetric cryptosystem chip, such as ICL's IPACRYPT 200 can easily exceed 10 M bits/second.

5. Data integrity

In the days before electronic communication, the integrity of a written letter could be assured by sealing it with an unforgeable wax imprint. If upon delivery of such a letter the seal was unbroken, the recipient could assume that the letter had not been tampered with.

The integrity of electronic information can be assured by a digital seal which is a function of the data and a secret key. When produced by symmetric techniques, either by an authenticator algorithm or a cipher, it is called a 'Message Authentication Code' (MAC). The sender of a message would append a seal, say of 128 bits, computed from its contents and a secret key. The receiver verifies this by computing another seal from the contents of the received message using the same key as the sender.

If the seals match it can be assumed that:

(a) neither the data nor the seal has been tampered with (Data Integrity);
(b) the message was sent by the only other entity to have knowledge of the key (Data Origin Authentication);

Figure 3 illustrates an example where a timestamp has been included in the message in order to detect replays of earlier transmissions.

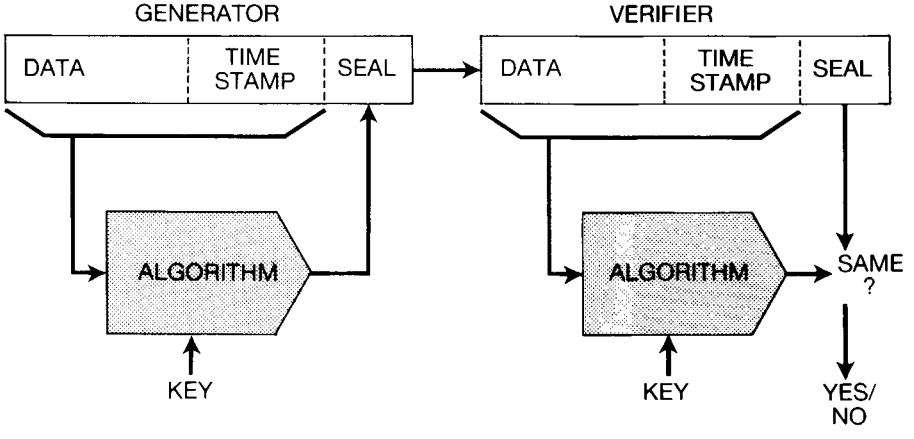


Figure 3 Use of a Symmetric Integrity Seal

A property of some Public Key Systems, including RSA, is that the encryption and decryption functions are commutative. That is, the decryption function can be applied to a message to obtain a transformation to which the encryption function can be applied in order to retrieve the original

message. Because such a transformation can only be generated by the holder of the secret key, it is called a 'Digital Signature'.

Figure 4 illustrates the basic principles of digital signatures, where the sender of a message also sends a digital signature produced by applying the decryption function on the message using his/her secret key. The receiver can compare the received message with the result of applying the encryption function to the signature using the public key of the sender.

If these match, the receiver can be confident that:

(a) the signed message has not been tampered with (Data Integrity);
(b) the message originated from the claimed entity and was not forged by somebody else (Data Origin Authentication).

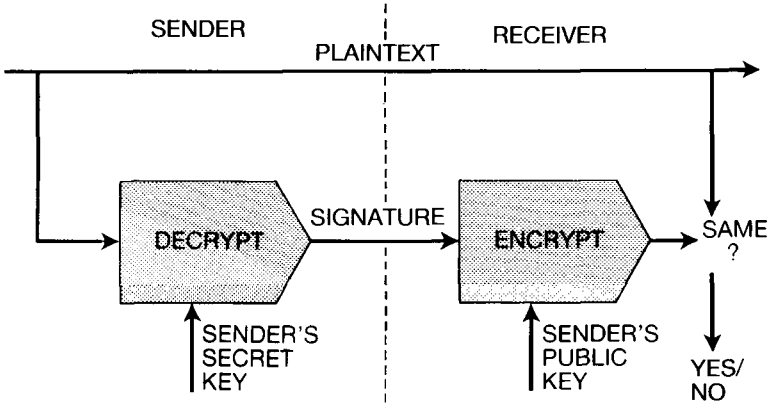


Figure 4 Principles of Digital Signatures

With a symmetric seal, the receiver shares a key with the sender and is therefore able to create valid seals on messages of its own choosing. The sender could deny sending a message by claiming that the receiver must have forged it.

With a digital signature, the sender cannot deny sending a message because the received signature could only have been generated by the sender (Non-Repudiation with Proof of Origin). In such a dispute, a third party arbitrator would be able to verify the signature on a received message using the public key of the alleged sender. Therefore in systems where non-repudiation is important (e.g. banking, finance, retail), a record must be kept of all signed messages as evidence in case of dispute.

Similarly, if the communications protocol always returns a signed acknowledgement to a received message, the receiver cannot deny having received it (Non-Repudiation with Proof of Delivery).

6. Digital signature standards

ISO are developing an International Standard to permit a small message to be recovered directly from its digital signature (ref. [4]). In this RSA-based scheme, the message to be signed must be less than half of the length of the RSA modulus (see Appendix). The size of the message is doubled by adding 4 redundancy bits (a nibble) for every nibble of data, the value being dependent upon the value of the data nibble. This is then signed under the sender's secret key.

The receiver of the signature applies the sender's public key to retrieve the padded message. If the correct redundancy is present, it is removed and the resulting message accepted as valid.

In order to sign a message which is longer than the block length of the cryptosystem, it is common practice to use a hash function in order to condense the message to a single block which is then signed and appended to the message. The receiver would compute a new hash from the received message and compare it with the hash obtained by encrypting the signature using the sender's public key. If they match, the message can be accepted as genuine. ISO will produce a standard for such a scheme after they have developed a standard for hash functions suitable for use with digital signatures (ref. [5]).

7. Security of digital signatures

Assuming that there are no flaws in the Public Key Cryptosystem used, the strength of a digital signature relies on the security of the secret key and the validity of the public key with which the signature is checked.

If another user can obtain somebody else's secret key, then he/she can forge signatures on messages which will verify as belonging to the original owner. The solution to this problem is to design a system where only the processor that performs the signing is allowed access to the secret key. For example, in a smart card the secret key could be stored on the same chip as the processor and not even the card holder would be allowed to see it (the smart card must also be made physically secure).

It is also important that the public key used to verify a signature is genuine. One way to ensure this is for a trusted 'Certification Authority' to certify a public key by placing its digital signature upon it. The authenticity of a public key can then be tested by checking the Certification Authority's signature on its certificate.

Before creating a certificate, the Certification Authority must be certain of the user's true identity. Besides containing the public key, each certificate would contain additional information such as the identity of the Certification Authority (there could be more than one), the identity of the user, the period of validity of the certificate and the algorithm used to sign the certificate.

8. Certificates and the OSI Directory

The OSI Directory (ISO 9594/CCITT X.500) is a proposed international standard for a distributed database which allows a user to register its public key certificate along with other communication details (network address, etc). These can be accessed via the Directory enquiry service by other users wishing to communicate with the user.

The Directory Authentication Framework (ref. [6]) allows certificates to be created by several Certification Authorities and defines how 'certification paths' can be set up to permit a user to authenticate a certificate created by a different Certification Authority.

Once a user's certificate has been authenticated, its public key can then be used to encrypt messages that are sent to that user or to verify digital signatures on messages received from that user.

Indeed this is the basis for a number of security services provided in the 1988 version of the CCITT Message Handling System X.400 (ISO 10021 Message Oriented Text Interchange System).

9. Certificate based Peer Authentication and Key Distribution

The Directory Authentication Framework also defines how communicating entities in the same OSI layer can mutually authenticate themselves using digital signatures. This will be superseded by a separate standard for Peer Entity Authentication. Currently drafts exist for standards for mechanisms using symmetric cryptography (ref. [7]) and using Public key cryptography with a two-way (ref. [8]) or three-way (ref. [9]) handshake. These will be combined into a single international standard. Only the two-way handshake mechanism will be described here for illustration.

In order for two entities with identities A and B to authenticate each other, the following steps are performed:

(1) A requests B's certificate from an Authentication Server (this could be the Directory enquiry service).
(2) The Authentication Server (AS) returns B's certificate to A.
(3) A validates B's public key by checking the signature on the certificate using the public key of the Certification Authority which created it.
(4) If the public key is valid, A sends to B its own certificate together with a 'token' containing B's identity, a time and date stamp (to guard against replays), and (optionally) some data protected by encryption under B's public key. The token is signed using A's secret key.
(5) B validates A's public key by checking the signature on its certificate and then uses it to check the validity of the token. If the token is valid, B accepts that it was sent by A. Any encrypted data is decrypted using B's secret key.

(6) B returns a new token to A containing A's identity, a time and date stamp and (optionally) data encrypted under A's public key. The token is signed using B's secret key.
(7) The returned token is validated by A by checking its signature using B's public key. If the token is valid, A accepts that it was sent by B. Any encrypted data is decrypted using A's secret key.

This is illustrated in figure 5.

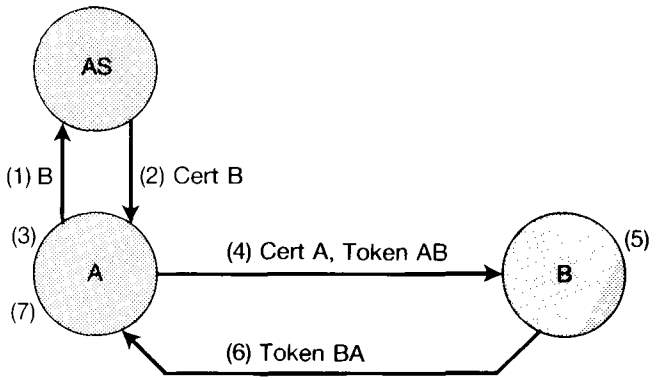


Figure 5 Peer Entity Authentication with a two-way Handshake

The tokens exchanged in a Peer Entity Authentication protocol can be used to carry data encrypted under the recipient's public key. This could be used to distribute a key for use by a symmetric cryptosystem. Alternatively, both entities could use these encrypted data fields to exchange keying information that could be used in the generation of keys (e.g. by the concatenation of random numbers).

The use of a certificate-based key distribution protocol has been adopted by the U.S. OSI security initiative, the Secure Data Network System (SDNS), and will probably be adopted by ICL for use in IPA security services.

10. Smart Cards and Intelligent Tokens

The most important application for digital signatures in the immediate future is in the replacement of the magnetic striped credit card by more secure 'Smart Cards'. These contain a microprocessor and memory and can be used to generate digital signatures on messages. In retail and banking applications, the digital signature generated by a customer's card on an electronic message would authorize a transaction in the same way as a hand written signature does today.

If a card is stolen, the thief can not use it unless he/she knows the customer's PIN which activates the device. The PIN and the secret key can not be read

out of the card except by reverse engineering, something that is out of the capabilities and price of most thieves. Fake cards will be easily detected because the signatures that they produce will not verify correctly against the public key registered for that card.

Other applications for smart cards using digital signatures include:

(a) The identification of users when they log on to a computer system or network, replacing the password which is far too often compromised by unauthorized users.
(b) The operation of electronic door locks in restricted areas of buildings.

In 1982, ICL was one of the founding members of the Tokens and Transactions Control Consortium (TTCC), under which the National Physical Laboratory developed an 'intelligent' token, a device more advanced than a smart card with alphanumeric display and keypad that can produce a RSA digital signature on a block of data.

In 1988, TTCC was disbanded and its successor the Advanced Tokens Technology Club (ATTC) launched, ICL again being a founder member. The principal aim of ATTC is to promote awareness of intelligent tokens and their applications in areas such as access control and secure transactions.

11. Summary

Although devised over ten years ago, it is only recently that technology has allowed the practical and efficient implementation of public key systems. Use of public key systems has been initially slow, but is now becoming widespread partly due to the emergence and acceptance of the smart card by financial institutions and by the development of security-related standards.

In this article, it has been shown how Public Key Systems can be used to provide the mechanisms upon which security services can be built. Public key systems are unlikely to replace symmetric cryptosystems for providing data confidentiality because they are comparatively slow. However, Public Key Systems will become widely used to provide authentication and non-repudiation mechanisms and in the distribution of keys for conventional cryptosystems.

ICL is committed to Open Systems standards and will adopt and in many cases contribute to the development of international Open Systems security standards. Internal ICL standards addressing the use of Public Key Systems already exist and several projects are under way which will implement and use Public Key Systems, particularly for the purposes of Data Origin Authentication, Non-repudiation, Peer Entity Authentication and Key Distribution.

References

1 ISO.TC97/SC21/WG1: 'ISO 7498/part 2 – Security architecture', 1987
2 DIFFIE, W. & HELLMAN, M. E.: 'New directions in cryptography', IEEE Transactions on Information Theory, 22(6), p. 644–654, 1976
3 RIVEST, R. L., SHAMIR, A. & ADLEMAN, L.: 'A method for obtaining digital signatures and public key cryptosystems', Communications of the ACM, 21(2), p. 120–126, 1978
4 ISO DP 9798 'A Digital Signature Scheme with Direct Message Recovery using a Public Key System' (formerly 'with Shadow')
5 ISO DP 10118 'Hash Functions for Digital Signatures'
6 ISO DIS 9594/8 'OSI – The Directory – Authentication Framework'
7 ISO DP 9798 'Peer Entity Authentication Mechanisms using an N-bit Secret Key Algorithm'
8 ISO DP 9799 'Peer Entity Authentication using a Public Key Algorithm with a Two-way Handshake'
9 ISO DP 10117 'Peer Entity Authentication Mechanism using a Public Key Algorithm with a Three-way Handshake'
10 GORDON, J.: 'Strong RSA Keys', IEE Electronic Letters, 20(12), p. 514–516, 1984

APPENDIX – The RSA Public Key Cryptosystem

The RSA Algorithm

The operation of RSA [ref. 3] is described below together with explanations of the mathematics which may be unfamiliar to the reader:

(a) The user generates two large but random prime numbers P and Q; these should be of the order of 150 decimal digits or more.

A 'Prime Number' is a number that has no factors except 1 and itself.

(b) The product of the prime numbers is computed $N = P*Q$, where '*' denotes multiplication.

(c) The user chooses an integer E such that it is 'relatively prime' to $(P-1)*(Q-1)$.

Two numbers X and Y are 'Relatively Prime' or 'Coprime' if they have no common divisor except 1, for example 16 (2*8, 4*4, 1*16) and 9 (3*3, 1*9).

(d) The user computes an integer D such that it is the 'multiplicative inverse' of E modulo $(P-1)*(Q-1)$.

X is the 'Multiplicative Inverse' of Y modulo Z if $X*Y = 1$ modulo Z. That is, the remainder after dividing $X*Y$ by Z is 1.
For example: let $X = 3$, $Y = 7$ and $Z = 10$.

$$3*7 = 21 = 1 \text{ modulo } 10 \text{ (since } 21/10 = 2 \text{ remainder } 1)$$

Thus 3 is the multiplicative inverse of 7 modulo 10.

(e) The pair of values E and N are announced as the user's public key, whilst D and N are kept as the secret key. The prime factors of N are kept secret to make factorization of N infeasible and thus prevent determining the value of D from E.

(f) A message, when treated as an integer M numerically less than N, is encrypted by raising it to the power of E (the encryption exponent) modulo N, that is the ciphertext C is the remainder when M^E is divided by N:

$$C = M^E \text{ modulo } N$$

(g) The user decrypts the ciphertext C by raising it to the power D (the decryption exponent) modulo N:

$$M = C^D \text{ modulo } N$$

RSA works due to Euler's theorem, which states that if M is coprime to N:

$$M^{Q(N)} = 1 \text{ modulo } N \quad (1)$$

$Q(N)$ is the Euler Totient function and equals the number of positive integers less than N that are coprime to N. If $N = P*Q$, the product of two primes, then $Q(N) = (P-1)*(Q-1)$.

The generation of the keys centres on: $E*D = 1$ modulo $Q(N)$.

Remembering that this result is the remainder of $E*D$ after division by $Q(N)$, then this expression can be re-written:

$$E*D = K*Q(N) + 1, \text{ for some integer } K. \quad (2)$$

The decryption function is
$$\begin{aligned} M &= C^D \text{ modulo } N \\ &= (M^E)^D \text{ modulo } N \\ &= M^{E*D} \text{ modulo } N \\ &= M^{K*Q(N)+1} \text{ modulo } N \text{ (from (2))} \\ &= M*(M^{K}{}_{*}{}^{Q(N)}) \text{ modulo } N \\ &= M*(M^{Q(N)})^K \text{ modulo } N \\ &= M*(1)^K \text{ modulo } N \quad \text{(from (1))} \\ &= M \text{ (since } M < N) \end{aligned}$$

Security of RSA

There are no known attacks on RSA that are faster than factoring the modulus. Currently numbers of between 90 and 100 decimal digits can be factored in a couple of weeks using a network of several dozen powerful workstations.

The most commonly used modulus length and the length recommended by ISO is 512 bits (about 154 decimal digits). This is currently considered 'safe'.

Unlike other Public Key cryptosystems, RSA is widely regarded as being both secure and practical providing the following guidelines for key parameters are adhered to:

1. To make RSA secure against some well known attacks, the prime factors P and Q must be 'strong' as defined by Gordon (ref. [10]) such that:

(i) P and Q are large randomly chosen primes;
(ii) $|P - Q|$ (the difference between P and Q) is large;
(iii) $P + 1$ has a large prime factor;
(iv) $Q + 1$ has a large prime factor;
(v) $P - 1$ has a large prime factor, R;
(vi) $Q - 1$ has a large prime factor, S;
(vii) $R - 1$ has a large prime factor;
(viii) $S - 1$ has a large prime factor;

2. Groups of users should not share a common modulus.
3. The encryption exponent should be greater than the length of the modulus in bits. That is, low exponents should be avoided, a standard exponent of 65537 ($2^{16} + 1$) has been adopted by the EFTPOS UK and TeleTrust/OSIS projects and recommended by ISO in the OSI Directory.

Security Classes and Access Rights in a Distributed System

R. W. Jones
ICL Defence Systems, International Computers Ltd., Eskdale Road, Wokingham, Reading, RG11 5TT, U.K.

Summary

A model is described for the management and use of a network of computing resources in which the control of security is to be explicit. It is intended to be useful in military and civil applications. A generic form for a security class is defined. Using this form a total security class for a particular system may be defined in terms of attributes with agreed meanings in the environment of the use of the system. The total class contains subclasses which are used to classify all the components of the system. A subclass is a class which is more specific than its containing class. For each class a set of access rights is defined. An access right is a set of operations which may be performed upon components classified by the class of the access right. Components include data items, computing resources and end users. An active component (e.g. a computing resource) is one which may operate upon other components. Each active component has a clearance which is a class with associated access rights. An access right is a set of operations which may be performed upon components classified by the class of the access right. A security policy is a set of rules which associates classification classes with maximum permitted clearances.

Communication between security domains is discussed in terms of the model, as is secure system construction. The model is related to other published models and standards.

The first version of this paper was presented at the Workshop on Data Integrity held at Gaithersburg, USA, in January 1989 and organised by the US National Institute of Standards and Technology. The present version incorporates changes made as a result of that presentation and commented on in a series of Notes at the end.

1. Introduction

1.1 Aims and summary of the paper

The paper describes a general design for a secure system which uses a potentially distributed network of computing resources. The design is based

on the idea of a *security class*, a generic form for which is defined in the paper. The generic form provides a notation which may be used to describe specific security classes for an individual system. The paper describes a model which applies both to the secure creation and management of a system and to its secure use. It uses and builds on ideas described in ref. 1 and ref. 2.

The aims of the paper are to:

(i) describe a general design for a secure system (covering civil and military needs) and the means of tailoring it to a particular system,
(ii) provide a notation which enables the requirements of a secure system to be related to its design,
(iii) clarify terminology and ideas in secure system design.

The paper is organised as follows.

Section 1.2 describes the context of the paper.
Section 2 defines some terms.
Section 3 defines a security class and explains its use.
Section 4 uses the idea of a security class to introduce access rights, classifications and clearances, these being used to describe how security relates to the components of a system.
Section 5 describes specific access rights which are useful in most systems. They include those needed for:
change of security class (applicable to identification and authentication),
creating and controlling the components of a system,
delegation of authority.
Section 6 compares the model of this paper with other published models and standards.
Section 7 provides examples in terms of the paper.
Section 8 defines security policies in terms of this paper.
Section 9 defines security domains in terms of this paper.
Section 10 describes secure system construction.
Section 11 draws conclusions and considers future work needed.

1.2 Background

In non-automated systems where people have to handle confidential information it is customary to give the information a classification, for example, 'company confidential'. The people are then also classified acording to the amount of trust that should be placed in them. This takes into account both the likelihood that they will abide by the rules which safeguard the information and their need to know it in order to perform their function properly. The people in such an environment, therefore, have both a classification, based on their own trustworthiness, and a clearance which describes the kinds of information they may handle.

There is a management function for the system which decides how information and people are to be classified, what these classifications mean, in terms of the characteristics of the people and the information and objects they manipulate, and which classes of people may have which clearances. The management also provides the classifications and clearances based on these rules and changes and deletes them as necessary. The amount of formality attached to this management function depends upon the environment. It may be very informal in a small commercial firm and is very formal in the armed forces (formal in the sense that the processes involved are performed according to explicit rules).

If we now consider that a network of computer resources is to be used as part of the system which handles information securely we see that we need mechanisms which perform the equivalent of both the management and the operational functions described above. They need not mimic them exactly but it must be clear that they allow the system to be created, managed and operated securely. If we provide the mechanisms which cause the system to operate securely but provide them without using centrally provided mechanisms, for example by building access control checks into application code rather than by installing them into supporting control software, we are in the position of the small firm mentioned above. Since the computer system itself allows us to describe rules precisely and ensure that they are obeyed we have the opportunity to make explicit both the provision and the operation of the security mechanisms. This paper attempts to contribute to this by describing a model on which mechanisms can be based which perform both the management and operational security functions.

The context, described more precisely, is a community of computer installations which communicate using telecommunications and agreed standard protocols (for example those of ref. 3). Those standards formalise a computer installation which communicates according to its standards as an 'open system'. They allow a single open system to support a number of software entities and allow any entity in an open system to exchange messages with an entity in another open system. This is illustrated in figure 1.

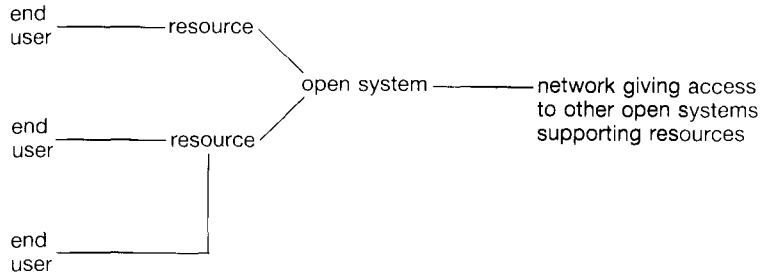


Fig. 1.

2. Terminology [See Note 1]

In order to discuss the model a set of terms is now defined. The aim is to depart from accepted usage only to avoid misunderstanding.

An **end user** is a person or entity which is not controlled by a resource of the network, but which can communicate with a resource. An end user may be, for example, a person sitting at a terminal.

A **resource** is a logical part of a computer system at one location which can store data items, which can communicate with other such resources or with end users and which has a current state which determines its actions.

A **data item** is an item of data which is created with a defined security class and the parts of which necessarily have the same class. The term message is used to mean a data item when it is sent from one resource to another.

A **channel** is the route by means of which a resource communicates with another resource or with an end user.

An **access right** is a collection of operations which is defined for a particular security class. The holder of an access right has permission to perform those operations upon components which are described by the class in question. Where there is no chance of confusion the term 'right' is used instead of 'access right'. There may be more than one access right for any given class in order to divide operations into groups and control access to groups individually.

A **component** is any entity which has a security class. End users, resources, data items, and channels, all have security classes and are therefore components.

Components may be **active** or **passive** depending on whether they can hold **access rights**. End users and resources and channels are active components. Data items and access rights are passive components. A channel may only hold access rights which enable it to transmit messages.

The components named above may be grouped diagramatically, therefore, as follows: [See Note 2]

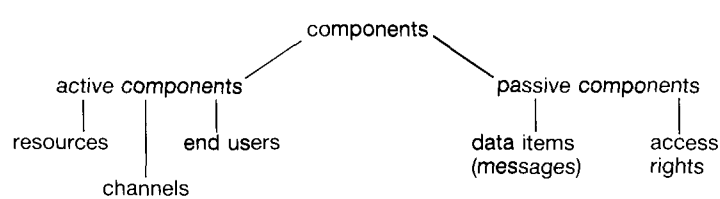


Fig. 2.

Refs 1 and 2 described a model in terms of rights which enabled access control rules to be enforced. The present paper builds on these concepts and adds to them the idea of **security classes**. These are compatible with and include the concepts used in data classifications and subject clearances in military systems and, with some modifications, in civil systems. Where there is no chance of confusion the term 'class' is used rather than 'security class'. An exact meaning of the term is given in section 3, which also introduces other terms for the purpose.

A **classification** is the security class of a component which determines whether or not it may *be operated on* by another component. Every component has a classification.

A **clearance** is the security class and associated access rights associated with an active component which is used in determining whether or not the component may *operate on* another object. An active component has a classification and a clearance. A passive component has a classification but no clearance. The relationships among a clearance, a classification and an access right are described in section 3.

The term '**system**' is used to mean a communicating set of components whose access to each other is controlled by mechanisms which take account of their security classes. There is no implication that the components are dedicated to work which is all interrelated, although that is likely to be true in practice.

The term '**subject**' is used to refer to an active component which performs an operation upon another component.

The term '**object**' is used to refer to a component (active or passive) upon which an operation is performed.

3. Security Classes [See Note 3]

If we consider once more the non automated secure environment we can see that one way of stating rules of access is to have a unique identity for each object to which access is to be controlled and to provide each person with a list of the objects he may access, together with, in each case, a list of the operations he may perform. In order to provide clearances to the people, they themselves are treated as objects and some manager has the list of people to whom he may give a clearance or whose clearance he may amend. The procedures needed, therefore, both to *operate* and to *manage* the secure system in this simple way are similar. For each person who may perform operations, a list of the permitted objects is needed and, for each of those, a list of the operations which may be performed. This applies both to operation and management.

In a more complicated system clearances are expressed in terms of attributes of the objects accessed for two reasons:

(i) it enables objects to be grouped so that clearances are more concise,
(ii) the attributes of the object's classification have meanings which have a relationship to the attributes of the accessor's classification (e.g. an object with the attributes 'payroll information' is to be available only to someone with the attribute 'member of the payroll department').

The definition of a security class which now follows is based on attributes. The meanings of individual attributes are not defined since they depend upon and are agreed for a particular system. An attribute may define an object uniquely or may describe one of its characteristics.

Extended **Backus-Naur** notation is used in the following definition, the meta-symbols having the following meanings: [See Note 4]

::= means 'is defined as',
| means 'or',
; terminates a syntactic rule
' angle quote symbols delimit items which appear as written in the rules defined
* indicates one or more occurrences of the item to its left.
a string of (possibly hyphenated) letters is an identifier which is either defined by one of the rules or is described informally

The generic form of a security class is then as follows:

```
class-definition::= class-name ':' lower-limit ',' qualifier*;
class-name::= name;
lower-limit::= integer;
qualifier::= class-name | attribute;
attribute::= name
```

where $1 \leqslant$ lower limit $\leqslant$ number of qualifiers.

An attribute is a name declared for the definition and control of the security of the system, as described above.

A name is a sequence of characters generated according to some rule that ensures that each name generated is distinct from all those generated previously to describe the system's security for the same purpose

A lower limit defines that at least that number of the defined qualifiers is present in the class being defined or in one of its subclasses.

Two particular values of 'lower limit' are worth distinguishing:

(i) the value 1, where the class describes a collection of qualifiers, any one or more of which may be present;

(ii) the value of 'lower limit' is equal to the number of qualifiers; this describes a class in which all the qualifiers are present. A single case of its use is where the qualifiers are attributes and describe all those which must be present to define the class.

A subclass is derived from the definition of a class, with the restriction that a subclass cannot be identical with the class from which it is derived [see Note 5]

(i) by selecting n of its qualifiers such that $b \leqslant n < q$, where
b is the lower limit of the class from which the selection is made and
q is the number of qualifiers in the class from which the selection is made.
and
(ii) by selecting a lower limit for the subclass such that $b \leqslant$ lower limit $\leqslant n$.

A subclass is also a class. If a class S may be derived from a class C then C will be called the **containing class** of S.

A subclass may also be derived by replacing one or more of the classes used in the definition of a class by a subclass: see Fig. 3 and the subsequent explanation.

The intention behind the definition of a class may be illustrated by a small example. Let us suppose that a commercial firm creates documents, access to which is to be controlled, depending on the trustworthiness and duties of the accessors. The attributes 'confidential', 'pay' and 'plans' are used to classify the documents so that security controls may be applied. Some documents about pay are confidential; some are not. Some documents about plans are confidential; some are not. Some documents are simply classified as confidential and these are not concerned with either pay or plans. The people who may use the documents are cleared in terms of the documents they may access as follows:

all documents,
all confidential documents,
confidential documents not concerned with pay or plans,
confidential documents about pay or plans or both,
only non-confidential pay documents,
only non-confidential plans.

The class needed to express these needs is as follows:

```
all: 1, all-conf, subjects;
all-conf: 1, conf, conf-subjects;
conf-subjects: 2, conf, subjects;
subjects: 1, pay, plans;
```

It is illustrated by the following graph:

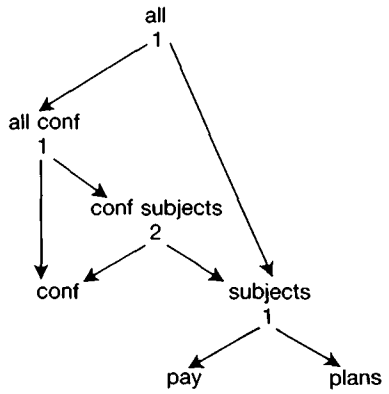


Fig. 3.

[The graph makes it clear that the figure 1 could be replaced in each case by the 'inclusive or' operation, meaning that one or more of the qualifiers must be present, and the figure 2 by the 'and' operation, meaning that all the qualifiers must be present. The generic form of the syntax could be rewritten to match this and would be trivially different. The forms where the lower limit lies between 1 and the number of qualifiers would be more cumbersome to express.]

It will be seen from the above example that the derivation of a subclass produces a more specific description in terms of security. The class 'conf-subjects' has three subclasses illustrated graphically below.

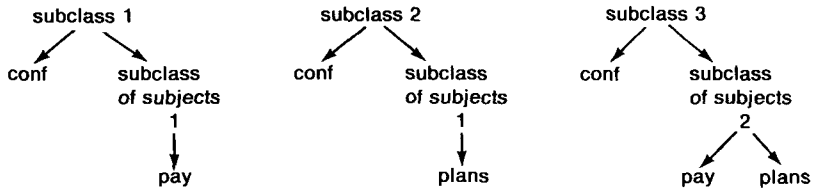


Fig. 4.

If a class from which subclasses may be derived is used to classify a component it means that the component concerns any one or more of those subclasses. It may be used also to describe the fact that an active component has a multi-class clearance. This is described further in section 4.

A selected subclass may be equivalent to one defined explicitly and used as a qualifier in the definition of the class from which the subclass derives, for example in selecting the class 'authorised' from the class 'accessors', where 'accessors' is defined as

accessors: 1, authorised, unauthorised;
where 'authorised' is defined as;

authorised: 1, Alice, Bob, Charles;
and 'unauthorised' is an attribute defined for the system.

It may implicitly define, for example, the class obtained by deriving the subclass '1, Alice, Bob' from 'authorised'.

A class is defined explicitly so that it may be used as a classification or clearance of a component and in order to be able to define access rights and operations for the class.

Two classes may contain the same subclass without themselves being identical or one of them containing the other, for example the classes

'1, Xusers, unauthorised' and '1, Yusers, unauthorised'.

4. Classifications, Clearances and Access Rights

A class is defined for the total system. Each class used in the system is a subset of this total class.

For each class of the system a set of access rights is declared. Each of these access rights defines one or more operations for that class. (In refs 1 and 2 a particular right applies to a single defined target resource. It is generalised here to apply to a class, which may, in a particular case, be a class which is used to classify only one component.)

Each component of the sysem has a classification which is a class. The classification defines the operations which may be performed upon the component (those of its class) and the kind of component it is in terms of security (because the attributes of its class have an agreed meaning). [See Note 6].

Each active component also has a clearance which is a class. The clearance defines the operations which the component may perform upon other components (those whose classifications or subclasses of whose classifications appear in its clearance). The clearance therefore defines the trust which is placed in the component which possesses it. This is illustrated in figure 5. [See Note 7].

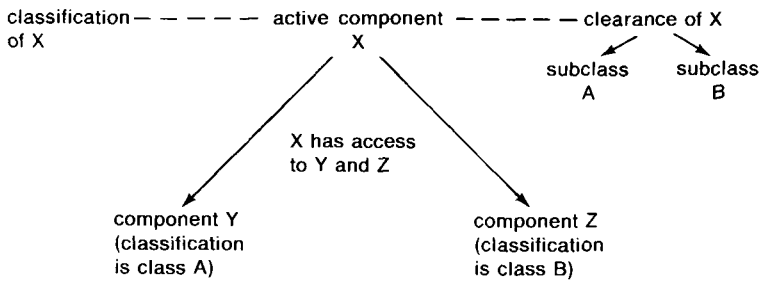


Fig. 5.

It is necessary to provide different accessors with different rights in respect of the same component accessed. Two methods of doing this are described here.

(i) The class used for the classification of the component to be accessed may be a subclass of more than one other class. Each of the containing classes has a set of rights which applies to its subclass. The union of these sets of rights form the total set which applies to a component classified by the subclass. This is illustrated in figure 6. [See Note 8].

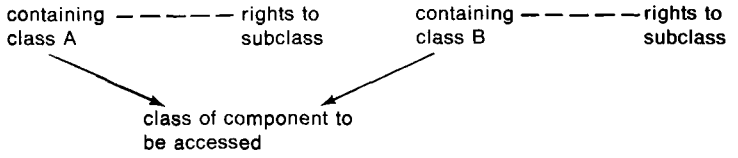


Fig. 6.

(In an implementation of the design operations other than the union of the sets of rights may be used for efficiency; for example, the subclass may define all the applicable rights and the containing classes those which are not available via that containing class).

(ii) The clearance of a resource defines for its most general class and for its subclasses individually the access rights which it is allowed to possess. These are the total set or a subset of those defined for the class. For some of its subclasses none may be allowed.

Notionally the second method is unnecessary since one may define a containing class with the selection of rights needed. It implies that a class may be created as needed and associated with the clearance. For the purposes of this paper it is assumed that both methods are available.

By analogy with the idea of providing a clearance with a subset of the rights defined for its class one might allow the classification of a component to have a subset of the set of rights defined for its class, thus restricting the operations available to all accessors. Again this is notionally unnecessary since a class may be defined with the required rights. It may be useful in practice to avoid a large number of classes.

Consider a class defined as follows:
accessors: 1, Alice, Bob, Charles, unauthorised;

There are four subclasses of interest, namely those defined by the single attributes Alice, Bob, Charles; and 'unauthorised'. Let us assume that the first three of these more elementary classes are used to classify data from three authorised users of the system and that 'unauthorised' is a class which describes data from any unauthorised person who tries to access the system. The resource which may be accessed by anyone who approaches the system has 'accessors' within its clearance. It receives data from a user who wishes to

use the system via a channel whose clearance is 'accessors'. The first message it receives therefore has the class 'accessors', representing the fact that it may come from any of those sources. The resource which is authorised to receive such a message then (in our example) engages in some procedure (authorised by one of its access rights) which allocates to the data received a subclass of 'accessors' (i.e. Alice, Bob Charles or 'unauthorised') representing the fact that it has authenticated one of the authorised users or recognised an attempt at a security breach.

There may be a resource between the accessors and the authorisation resource which is used to relay the accessors' messages but which is not authorised to classify the messages as coming from a particular user or as being unauthorised. This relay resource like the channel which carries the messages, has the class 'accessors' withing its clearance but has no rights which apply only to the subclasses (see figure 7).

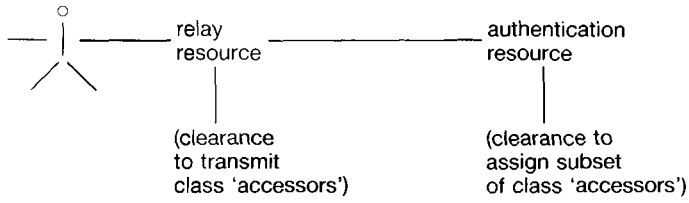


Fig. 7.

Thus, by defining the most general class for which access rights may be possessed and by defining rights for that class and its subclasses a clearance may restrict its owner in both the generality and the particularity of the classes which it may handle.

When a resource is created it is given a classification and a clearance as part of the creation operation. The clearance must have the following properties.

(i) It must conform to the security policy (i.e. it must be allowed for a resource of that classification, see section 8). It may be less than the permitted clearance in terms of classes and/or access rights.
(ii) It must not contain class/access right combinations not possessed by its creator.

A clearance shows the access rights which the holder of the clearance is *allowed* to possess for each of the classes in the clearance. In order to use an access right it must actually possess it. The right must be provided, either by the resource's creator or by some other resource with the right to do so (see section 5.2). [See Note 9].

These two features: allowing a clearance to be less than that imposed by the security policy and withholding the use of rights until they are supplied,

allow local control of security subject to an overriding central policy. Since the ability to create a resource and provide it with a clearance may be inherited a hierarchy of control over the assignment of clearances is possible. At any level a creator may pass on a clearance which is less permissive than the most permissive which it is allowed to pass on.

If a clearance contains conditions which depend upon the state of the environment (for example that an access right is available only at certain times of the day) this is not catered for by the model as described so far. This kind of condition may be fitted into the general design by associating an attribute of a class in a clearance with a procedure, whose result must be compatible with the corresponding attribute in the classification of the component accessed.

5. Specific Access Rights

In general the access rights and the operations which they provide are system specific and not defined by the model. Some, however, are generally useful and are described here.

5.1 Rights which apply to data items

Right to change classification
This right provides an operation to change the classification of a component to a subclass or containing class of its current class. An example of change to become a subclass has been given in Section 4. An example of change to become a containing class occurs when a resource sends a message. The classification of the message must be the class of a very general class in which sensitive data is indistinguishable from any other (in practice because the procedure which generalises the data involves encipherment).

5.2 Rights which apply to resources and end users

These are rights which enable resources to be created and controlled and which enable resources and end users to be provided with rights and to be sent data. They are very similar to the rights described in refs 1 and 2, modified to allow for the introduction of security classes. They are as follows.

Right to control resources
The right provides the following operation.

create *resource*	this creates a resource of the designated class and with the designated clearance.
activate	this makes the resource available to holders of rights to that class other than 'control';
suspend	this makes the resource unavailable except to the holder of its control right;
change	this changes the resources code;
delete	this withdraws the resource from service.

Right to authorise resources and end users [*See Note 10*]
The right provides the following operations.

supply This supplies to the target component a right held by the supplier; when the operation has been successfully performed both the supplier and the recipient possess the right; the right supplied must be for a class within the clearance of the target component. The supplier must be a resource.

withdraw This withdraws from the target component a right previously supplied by the same resource.

Right to send messages to active components
This provides the following operation.

send message This sends a message to a resource of the appropriate class. The message may instruct the resource to perform an operation upon data, in which case the sender must have clearance for the security classification of the data.

5.3 Rights which apply to Classes

Right to define classes
This provides the following operations

define class this defines the syntax of the class for which the right is held. For a particular class only one active component may hold the right.

create attribute this creates a new attribute by generating a name. The attribute may be used when defining a class.

This right enables local dynamically defined classes to be created. An example is where there is a need to create a file and exercise local control over access to it by other resources. A class may be defined for this purpose, whose only member is the file. In a more complicated case a resource may need to create several resources and enable each of them to create files and control access to them. This is achieved by passing on the right to define and create components classified by particular subclasses defined for the purpose.

6 Comparison with other Models and Standards for Secure Systems

6.1 Comparison with the Lattice Model of Information Flow

There are a number of significant differences between the model described here and the lattice model described in ref. 4. They are as follows.

(i) A lower limit to the number of attributes of a class is stated in the model described here. This enables a clearance to be given for a class which is defined by a particular set of attributes without providing clearance for the classes which are defined by fewer of those same attributes. Thus clearance for the class '2, secret, pay' does not provide clearance for things described as just 'secret'; only for those which also relate to pay.

(ii) Particular rights are defined for individual subclasses in the present model. They may be null or restricted in some clearances, to restrict the power of the component.
(iii) In the present model a classification is distinguished from a clearance.

6.2 Comparison with the Clark-Wilson Model

Ref. 5 describes a model for integrity. It has Constrained Data Items (CDIs) which are operated on by Integrity Verification Procedures (IVPs) and Transformation Procedures (TPs). The central ideas are:

(i) that data items whose integrity is to be protected are labelled as Constrained Data Items (CDIs),
(ii) that all CDIs are confirmed as conforming to a defined integrity specification by running Integrity Verification Procedures (IVPs),
(iii) that any CDI can be operated on to change its state only by a Transformation Procedure (TP) which is certified to be valid for that CDI,
(iv) that users, who must be authorised, are constrained to use only specified TPs on specified CDIs.

Rules are defined to ensure that the mechanisms and programs needed are certified as valid, that users are authenticated and that records are kept of the operation of TPs.

The model described in this paper provides a basic framework, using which a system which follows the rules of ref. 5 can be constructed. This is explained as follows.

(i) A CDI is a classification of a data item in order to specify and constrain the operations which may be performed upon it and corresponds to the classification of a data item using a security class.
(ii) An IVP is a particular kind of operation in terms of the model described here which should be controlled by an access right which is available only to components with the correct clearance. It is a particular feature of commercial systems that data is validated as a separate operation before it is operated upon. This is not a fundamental requirement of the model described here but the framework within which it can be done is provided.
(iii) A TP is an operation, in terms of the model described here, which, like an IVP, is controlled by an access right. A system which is to preserve data integrity must insist that operations are performed, not only on data of the correct kind but in the right order. This may be done, using the model described here, by using a qualifier of the class which classifies the data to prescribe the sequence in the following way.

Let us define a class as follows:

```
CDIX: 2, X, sequence
sequence: 1. unchecked, IVP, TPX1, TPX2, TPX3;
```

where X, let us say, means that the data belongs to project X and 'sequence' is used to ensure that operations are performed in the right order. 'X', 'IVP', 'TPX1', 'TPX2' and 'TPX3' are attributes of the class which describes the total system security.

An authorised user has a right to operate upon the subclass '2 X, unchecked'. The only operation the right allows him is one which performs the IVP operation. Part of the effect of this operation is to change the class of the data to a new subclass of 'CDIX', i.e. '2, X, IVP'. The user has a right to this class which enables the first appropriate TP to be performed. This similarly changes the subclass to '2, X TPX1'. In this manner the operations are performed in the correct sequence. It may be noted that the correct order depends upon the individual TPs behaving correctly. They operate in an environment in which the whole class 'CDIX' is valid. If there is a risk that the user of the managerial function which has the right to provide the TPs will collude with the user the operations may be located in more than one resource and the resources managed independently.

(iv) Users are constrained to use only specified TPs by providing a user with a classification and a clearance. The compatibility of the classification and the clearance are checked using the security policy (see section 8).

6.3 Relationship to the ECMA TC32/TG9 Security Model [See Note 11]

The ECMA TC32/TG9 model is described in ref. 6. It is a long document and it is not proposed here to explore in detail how it relates to the model of this paper. However, a set of security facilities are central to the ECMA model. This section therefore discusses them and relates them to the concepts described here. The facilities, with comments on them are as follows.

(i) *Subject sponsor facility*
This sponsors the human user to the secure system during authentication and monitors his subsequent activities. There is no distinguished separate entity in the model of this paper which corresponds to this. A resource may be created by any resource which has that right. The subject sponsor is a resource fundamental to the type of system described by ref. 6, which is, in this respect, a particular type of system which may be built using the model described here.

(ii) *Authentication facility*
This authenticates human users and applications of the system. It is represented in this model by resources which have a clearance which enables them to change the classification of a message to the subclass which identifies an individual user. In this model there is no insistence either that a single resource should authenticate all users or that there be a class which defines an individual user. Such requirements are essential for many systems and are not precluded by the model.

(iii) *Association management facility*
This sets up and maintains a secure association between entities of a secure distributed system which exchange information. In terms of this paper it is part of the functionality which is implied by the acknowledgement that the resources of the system may be distributed.

(iv) *Security state facility*
This records the current state of the system which is relevant to its security. In terms of this paper it is part of the functionality which is implied to ensure that a system operates correctly according to the model.

(v) *Security attribute facility*
This records the security attributes assigned to entities of the system. In terms of this paper its functionality is implied by the classifications and clearances of the components.

(vi) *Access Control facility*
This associates attributes with entities and also uses them to check whether an access request is to be granted. In terms of this paper the first of these functions is achieved by creating a resource with a particular classification and clearance. The second function is implied when any access is attempted.

(vii) *Inter-domain facility*
This controls access between entities in different security domains. Security domains are defined in terms of this paper in section 9 (q.v.).

(viii) *Security audit facility*
This records information about the use of the security functions of the system. This function is not explicit in the model of this paper. The model makes explicit all operations relevant to security classes and access rights. A security audit is provided by insisting that such operations are recorded, as is necessary in a practical system.

(ix) *Security recovery facility*
This facility is to enable a security administrator to take corrective action in case of a suspected breach of security. The model of this paper has nothing explicit to say about this. The assumption is that the required functionality is part of the definition of the individual system, which is built using the model described.

(x) *Cryptographic support facility*
This provides the cryptographic functions needed for the operation of the system. In terms of this paper it is a possible mechanism for implementing the model.

In general the model of ref. 6 is a prescription of facilities which are needed to provide a secure system of a particular kind which is likely to be frequently

needed. The model of this paper is more general in that it includes systems which would not conform to ref. 6.

The model of this paper is more abstract than ref. 6 in that it does not prescribe how the functionality needed should be provided. It is intended that such a prescription should be given separately.

6.4 Relationship to other standards and guidelines for security

Ref. 7, the US Department of Defense 'orange book', describes 'a uniform set of basic requirements and evaluation classes for assessing the effectiveness of security controls built into Automatic Data Processing (ADP) systems' (ref. 7 foreword). The model described here aims to aid system design to provide some of its requirements in an explicit manner. It is concerned in particular with mandatory and discretionary access control in the terms of that document.

Ref. 8 describes what is called a security architecture for the ISO reference model for open systems interconnection. It is therefore worth exploring how a model such as the one described here relates to it. The concerns of ref. 8 are:

(i) to describe appropriate security services and mechanisms,
(ii) to define where they may be provided in the reference model.

The security services described are grouped under the headings:

authentication,
access control,
data confidentiality,
data integrity,
non-repudiation.

All of these except the last relate directly to the model of this paper in that the open system interconnection standards may be used to provide communication services between remote resources. The security services may then be used to ensure that the communication is secure. The non-repudiation service has no direct relationship to the model. Its functions are directly relevant to system users in appropriate cases.

7 Examples of Use of Classifications, Clearances and Access Rights

Example 1 A class where all qualifiers are obligatory

Take the class defined as follows:

new product: 2, confidential, cars;

We may imagine that information of this class concerns a new product which is confidential and relates to cars. Now a resource which had only the clearance 'conf', defined as:

conf: 1, confidential:

or 'vehicles', defined as:

vehicles: cars, buses;

would be unable to hold an access right to the information.

A resource which had the clearance 'trusty', defined as:

trusty: 1, confidential, cars; or

trusty: 1, confidential, vehicles;

with ⟨vehicles⟩ defined as above, would be able to handle information which was classified by either or both of the attributes, provided it held the appropriate access rights.

A resource which had the clearance 'new product' would be able, with appropriate access rights, to handle information about the new product but not other information concerned with vehicles or other confidential information.

Example 2 A class where all qualifiers are not obligatory

A data item is recorded in the system and, to be recognised as genuine, must bear the signature of at least two of five possible signatories. Asume that there is a resource to which such messages are sent which is able to decide if the message is properly signed. This resource has, as part of its clearance, the class 'signed or unsigned', defined as:

signed or unsigned: 1, signed, unsigned;
signed: 2, Alice, Bob, Charles, Don, Eliza:

where the people's names are the attributes used to classify the signatories and 'unsigned' is a class with a single attribute, used to record that a message is not properly signed (including the case where it has only one genuine signature). One of the access rights possessed by the resource which receives the message permits it to perform an operation which classifies the message as 'signed' or 'unsigned'.

Example 3 A class where only one qualifier is obligatory

This is provided by the example in section 4 of a resource which communicates with end users in order to authenticate them so that they may then access parts of the system for which they are authorised. It must have within its clearance at least the classes of the channels by means of which it is accessed by the end users and, therefore, of the end users themselves. One of its rights is the ability to assign to a message it receives a particular subclass, which defines an end user.

Example 4 Classification of End Users

An end user may be given a classification which has a class with a single attribute which uniquely defines the end user. He (she or it) communicates

via a channel which has the end user class within its clearance. The channel has at least one other subclass to acknowledge the fact that the accessor may not be recognised as the genuine end user. The end user's clearance consists of the class which classifies the resource with which the end user communicates.

Example 5 Registered 'Users'

A likely design for a secure system that it records the clearance of each individual user and ensures that an end user, when authenticated, can access the parts of the system and the data for which he is cleared and nothing else. Using the model described here the information about such a registered user is a resource with a clearance which represents that of the user. The resource accessed initially by the end user (or some other resource or resources with which that in turn communicates according to the security rules of the system) is trusted to perform the correct mapping between the end user with his authenticated class and the class of the resource representing his use of the system. There is not necessarily a one to one mapping. On the one hand an end user may have more than one role to play in the system. On the other hand several end users may perform the same work at different times, for example in a system where shift work is needed.

Example 6 Hierarchical Classifications

In military security a document is given a classification of 'unclassified', 'restricted', 'confidential', 'secret' etc. A person who may read such a document has a clearance, for example of 'confidential', which allows him, if there are no other restrictions, to read documents whose classification is equal to or less than his clearance (in this case the documents classified as 'confidential' or 'restricted' or 'unclassified'). Using the model described here this clearance is described by defining a class to represent it thus:

confidential-clearance: 1, confidential, restricted, unclassified;

where 'confidential' etc. are attributes of the system. It is, of course possible to define classes that are not useful or which encourage insecurity, for example combining 'secret' with 'unclassified' and omitting the intervening ones. For this reason and for efficiency it is likely that the concept of hierarchies would be built into a practical system.

8 Security Policies

In terms of this model the security policy of a system is defined as follows. First define the total class of the system with each of its subclasses. For each class thus defined state the classes of the system with accompanying access rights, which may appear in the clearance of a component so classified.

This statement gives the maximum allowed clearance for any component. If a class is to classify only a passive component its maximum allowed clearance will be null. The total collection of maximum clearances for all classes is the security policy.

An active component may be created with less than its maximum allowed clearance in terms of classes, access rights or both. Since a component may not create another with a clearance more powerful than it possesses itself, this provides a way of creating locally security policies which are successively more stringent.

A security policy stated in this form is meaningful if the attributes used to define the classes of the system have a real meaning in terms of protection. Thus for example, one might suppose that a resource class defined as '2, class-A1, secure-location' might be allowed a powerful clearance. There is a very real difficulty in pinning down the needed attributes and their meanings which is beyond the scope of this paper. Moreover, since the meanings given to the term 'security policy' vary and are not always well defined it would be rash to predict that the definition given here will cover all of them. It is proposed as a tool for discussing security needs and consequent design.

A security policy can thus be represented diagrammatically as follows:

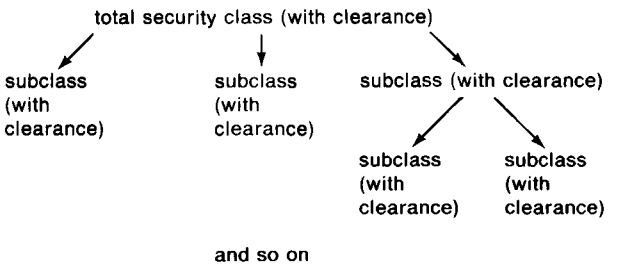


Fig. 8.

9 Security Domains

Section 8 described a security policy in terms of the ideas of this paper. When discussing secure distributed systems the term 'security domain' is often used and is usually equated with a collection of communicating entities which are subject to the same security policy. In the terms of this paper a security domain is characterised by its security class and its security policy. This tells us what kind of domain it is. To identify it uniquely we need to identify its creator and, if necessary, the name given to it by its creator to distinguish it from others.

We may now consider the possibility of communication between entities which are in different domains. We may assume that there is a practical need to do this if we consider that two different commercial organisations may set up secure automated systems separately and may then develop the need to communicate.

Let us assume that, in each case, a total class for the secure domain has been defined, based on a set of attributes with a defined meaning in the

environment to be automated. For each class and subclass access rights are defined which allow operations which are meaningful in the domain, either in terms of software available to the resources or operations performed by trusted personnel. For each class and subclass which can classify an active component the maximum allowed clearance is defined. Now, for interdomain communication a check must be performed that the clearance of the subject in one domain is, in some sense, compatible with classification of the object in the other domain.

For the sake of simplicity let us assume that all communication between the two domains passes through a single entity, which will be called the interdomain gateway (or simply the gateway when that is unambiguous). In each of the domains the gateway has a classification which makes use of the attribute 'gateway'. Each resource which is allowed to communicate with a resource in the other domain has, as one of the subclasses in its clearance, the class which classifies the gateway and at least the access right which enables it to pass data to the gateway. Conversely the gateway has a clearance for each domain which enables it to access the resources which may take part in interdomain communication. This is illustrated in figure 6.
The clearance of resource X to access other components is in terms of the classes of domain 1. To enable X to access components in domain 2 the gateway must contain information on the equivalence of classes and access rights in the two domains. The simplest arrangement is a one to one equivalence between a class/access right in domain 1 and a class/access right in domain 2. There need not be an equivalence in each case. Thus there may be classes in domain 1 which are inaccessible from domain 2 and vice versa. Similarly not all of the access rights may be made available to the other domain (e.g. information may be read but not changed from the other domain).

It is conceivable that the gateway may need to recognise an equivalence between class/access right combinations in the two domains where there is not a simple one to one relationship. This is beyond the scope of this paper.

Thus the gateway must hold a table of equivalences which it consults, together with the clearance of the would-be accessor, when access is attempted. The table is agreed and installed by collaboration of the management of the two domains. Secure domain construction and management is discussed in section 10.

10 Secure System Construction

Reference 1 described the secure construction of a system, starting with a completely trusted 'management entity' which had the power to create other entities and to devolve rights to them. It did not deal in classes (or therefore in classifications or clearances as described here). This section describes a similar process to that of ref. 1, making use of these additional ideas. As a preliminary it considers what it gained by the additions.

In the description in ref. 1 and here the starting 'management entity' which creates a distributed system is an organisation of people which is trusted to behave as a single entity to create those separate parts of the system which cannot be created by entirely automated means. The people perform procedures which correspond to the operations later performed by automatic entities to produce resources under their control and distribute rights among them. An elaboration of the concepts described in this paper is needed to describe in detail the operations to be performed both by the trusted people and by the automatic entities which they create. The comments made in this section therefore apply to procedures both by people and by automated resources.

The system of construction of ref. 1 provides the starting resource ('the management entity') with the ability to:

(i) create other resources,
(ii) provide them with rights, including the right to create other resources and to assign rights to them in their turn.

This produces a control hierarchy with which may be associated a complete record of which resources assign and use all rights in the system. However it lacks explicitly defined rules which state what rights may be assigned to particular resources. The introduction of the notion of a class enables a security policy to be defined in terms of the class and subclasses of the system being constructed and their access rights. When any resource is created it must be given a classification. Its maximum possible clearance is therefore defined by the security policy and an attempt to give it a clearance which does not accord with it is disallowed. In addition, each end user of the system has a classification and a clearance which derive from the classes of information he or she is allowed to receive from and send to the system and the operations to be allowed. Since an explicit classification is given to the resource which an end user may access a check is made when the end user receives access rights that his clearance matches the classification of the system he is allowed to access.

10.1 Creation of a Single Secure Domain

(i) A total class is defined for the domain which is to be created. For the class and each of its subclasses a set of access rights is defined and for each access right a set of operations.
(ii) A security policy is defined for the domain. It defines for each class (and subclass)/access right combination of the domain the attributes which must exist in the classification of the component which is cleared for that combination.
(iii) The management entity for the new domain creates the resource which it is to control directly, assigning to each of them a classification and a clearance. In each case the compatibility of the classification and clearance is automatically checked against the security policy. In the case of a distributed system these directly controlled resources are

those which exercise local control at individual locations. There are some resources (called basic resources in ref. 1) which are controlled by the management entity and which exist before it has created any resources of the domain. These correspond to the hardware and software to be used and the newly created resources are given access rights to them as appropriate. Their clearance to receive these rights is checked automatically against the security policy. The classification of these basic resources is decided by the management entity and is part of the basic decision making on which the security of the system ultimately depends. The attributes used to classify the basic resources reflect the judgement of the management entity of the security features needed in and provided by the equipment for the real world environment in which the security domain is used.

(iv) The directly controlled resources, thus created, create resources of their own and assign rights to their end users and to each other, as appropriate. These operations are automatically checked against the security policy.

10.2 Establishment of Secure Communications between Two Domains

Let us say that domains X and Y are to communicate according to mutually agreed rules of security. Then the following actions take place.

(i) The management entities of X and Y agree a correspondence of those of their class/access right combinations which are to be used for intercommunication.

(ii) The gateway between the two domains is formally created by both management entities and the table of correspondences is provided to it.

(iii) Within each domain separately access rights are provided to the gateway and appropriate other resources so that interdomain communication can take place.

11. Practical Considerations and Conclusions

The model described in this paper has been developed bearing in mind the following principles.

(i) Mechanisms which enforce security should be explicit in the system and separate from other functionality (e.g. from application code and system code which does not enforce security). There are a number of motives for this. It is more likely to be right, it is easier to change for those authorised, it may be made more difficult to change for those unauthorised, it is easier to check.

(ii) The security enforcement mechanisms should be of as general an application as is necessary. This has affected the model in three ways:
   (a) the model is intended to apply to both military and civil applications and is therefore a superset of the features normally considered in relation to military systems;
   (b) it unifies some of the concepts of secure systems which have been

elsewhere considered separately; authentication is treated as an authorised change of classification;

(c) the model and the mechanisms which derive from it apply both to the construction and management of a secure system and to its subsequent use.

It is hoped to use the model both as the basis of the design of secure systems and as a means of relating security requirements to system design. The most immediate tasks are seen as a more rigorous description of the model and an assessment of its usefulness by comparing it with practical systems.

Notes

As a result of the presentation of the original version of this paper at the Gaithersburg Workshop on Data Integrity, January 1989, and of subsequent conversations with a number of people, I have become aware of some mistakes which impeded understanding of the ideas in the paper. I have therefore changed it in some places and I describe these changes in the following notes. There are some cases also where comments might have caused me to describe things differently in particular to use different terminology, but where I have left the text as it was, thinking that a change might be confusing to readers of the first version; I give notes on such cases also. There is also some text here which I omitted from the version submitted to the Workshop, to shorten the paper.

1. (Section 2). The terms 'security class', 'classification' and 'clearance' have been criticised as having military connotations. I have not changed them. The term 'security category' has been suggested instead of 'security class' and 'remit' instead of 'clearance'. In retrospect I believe I should have avoided the term 'classification' or any substitute and described a component as *being* of a security category (which determines the operations which may be performed upon it) and as *having* a clearance (or remit) which determines what it may do).
2. (Section 2). In the original paper an access right was described as a component. It no longer is, as this seems clearer.
3. (Section 3). Some people have commented that the model I describe calls for capabilities to describe access rights, but not access lists. There is no such intention. The model is intended to be understood at a more abstract level. Both capabilities and access rights are possible mechanisms for representing rights in an implementation.
4. (Section 3). In the original paper I tried to use Extended BNF to describe both the generic form of a security class and for specific security classes and their relationship to each other. Some mistakes crept in and, therefore, some misunderstanding. In the version of the paper printed here I have tried to prevent the misunderstanding by using BNF only for the first purpose and correcting the mistakes.
5. (Section 3). Some of the points made later in the paper could have been put more simply if the term 'subclass' had been defined to include the containing class.
6. (Section 4). It is not intended that the mechanisms described here be the only means of distinguishing individual components, rather that they be used to distinguish sets of components which need to be distinguished for reasons of security.
7. (Section 4). If access is to be allowed the following must be true:

   (i) there must be a class which is contained in both the accessor's clearance and the target component's classification,

   (ii) that identified class must have rights which are contained in the accessor's clearance and they must have been supplied to the accessor and not subsequently withdrawn (see Section 5.2).

8. (Section 4). It is not clear in the paper when rights defined for a class should apply to its subclasses and vice versa. I believe it is necessary to be able to define rights which apply strictly to a class or to defined subclasses or to a class and its subclasses. More consideration is needed of this.

9. (Section 4). One criticism made of the paper is that it is not obvious that a right which appears in a clearance should also need to be 'supplied'. The motive for the extra flexibility is to enable components other than the original creator of a resource to provide and withdraw rights, subject to predefined constraints. There is an analogy with mandatory and discretionary access control in military systems.
10. (Section 5.2). It may be useful to be able to authorise a *class* of components.
11. (Section 6.3). This section did not appear in the paper as presented at Gaithersburg.

References

1 JONES, R. W.: 'The Design of Distributed Secure Logical Machines.', ICL Technical Journal, 1985 5(2).
2 JONES, R. W.: 'The Creation and Use of Explicit Rights in a Distributed System.', Proceedings of the Fourth IFIP Conference on Information Systems Security.
3 International Standard ISO 7498. Information Processing Systems — Open Systems Interconnection — Basic Reference Model.
4 DENNING, D. E. R.: 'Cryptography and Data Security'. Addison-Wesley, 1982.
5 CLARK, D. D. and WILSON, D. R.: 'A Comparison of Commercial and Military Computer Security Policies.', IEEE Computer Security Conference, 1987.
6 ECMA/TC32-TG9/87/60. Security Framework for the Application Layer of Open Systems.
7 Department of Defense Trusted Computer System Evaluation Criteria. National Computer Security Center, July 1986.
8 International Standard ISO 7498/2. Security Architecture.

KNOWLEDGE ENGINEERING

Building a Marketeer's Workbench: an expert system applied to the Marketing Planning Process

Stephen Aitken
ICL Marketing Development & Training Wokefield Park, Reading, Berkshire
Harry Bintley
ICL Knowledge Engineering, Manchester

Abstract

The paper describes the planning, design, development and introduction of an expert system to support the marketing process in ICL; this is part of an overall project to build a Marketeer's Workbench. The aim of the expert system is to improve the quality and productivity of the marketing planning carried out by ICL's marketing managers. It directly supports the company's planning procedures and complements the marketing training courses. The paper is in two parts: the first (section 1) describes the requirements for and use of the expert system, as seen by the marketeers; the second (sections 2–5), the design of the system, as seen by the developer.

1 Requirements for, and use of, the expert system

1.1 Introduction

In 1982 ICL introduced a new programme of marketing training courses, the aim of which was to introduce some new concepts and ways of working, together with a new vocabulary of marketing terminology. In outline these courses taught marketeers how to:

- identify opportunities from a scan of the environment
- define market segments
- project market and product life cycles
- develop a marketing mix over the life cycle
- produce competitive marketing strategies.

They included relatively detailed and complex techniques, such as the use of "perceptual maps", the "efficiency frontier" and "Differential Resource Analysis" (DRA) to arrive at competitive pricing and other strategies and to predict market share. All these concepts and processes were documented and explained, with examples, in the ICL Marketing Handbook.

In ICL, pricing strategies and world selling prices are determined in the corporate Business Divisions and pricing tactics are implemented in the countries around the world by the Sales Divisions; whilst the broad commercial policy is determined by Group Commercial. In the spring of 1987 Group Commercial asked Marketing Development and Training (MD&T) to re-examine the pricing course, as few of the marketeers, who had been through the training, were using the new techniques when presenting their rationale for prices for new products. The problem lay in the way the marketeers presented and discussed the case for their prices, and there was concern over whether Business Division marketeers had successfully taken account of the different competitive conditions prevailing in different countries. This pricing concerned both individual products and services and systems forming integrated business solutions.

1.2 Investigating the requirements

The initial reaction in MD&T was that it should develop or enhance the existing training course so as to have more impact on the participants. However, applying the marketing techniques that it teaches, it decided to carry out some market research among the ICL marketing and pricing community to determine how pricing was carried out; and during the summer of 1987 had a student interview a number of marketing and commercial managers across the company, both in Business Divisions and in the Countries.

The results of this and other studies showed:

- the pricing process taught in the training was seen as great in theory but not in practice. Those who had to carry it out did not always have the time or the research techniques to discover and predict the decision factors that buyers would use to evaluate products from competing suppliers. Even if they did have this information the calculations required to determine price were neither simple nor quick
- it was difficult for the marketeers in Business Divisions to obtain competitive data, but relatively easy for those in the Countries
- Business Divisions received insufficient feedback from Countries on the tactical implementation of their pricing policies
- after a product's introduction, prices were reviewed regularly in Countries but rarely in Business Divisions
- not all marketeers had pricing responsibility. In some areas a senior marketeer had taken on the role or had been appointed to control and coordinate the pricing and commercial policy for a group of people or products. This was usually in order to ensure consistency in pricing a range of products
- pricing was not always managed by only marketeers. In many Countries the whole management team was involved in regular price reviews covering new and existing products and services.

This market research activity provided a good statement of the target market segment for the new training course; that is:

- the people who needed training: not just marketeers
- how many there were, and where they were located
- how they worked: the problems they faced in obtaining information about their target market segments and about the competition, and what they did to determine prices
- how often they were involved in pricing decisions
- who would buy the new training for them and why, and how loyal they were to their current ways of pricing.

This definition of the market segment was put to a Human Factors Workshop where the marketeer presented his findings to designers, to a technical author and to Group Commercial. It was developed by the workshop, the end users being described in more detail and also the things they handled and the work they did. For example, three types of user were identified: primary users who set prices, secondary users who ratify prices and tertiary users who have a vested interest in prices. Thus the designer became immersed in detailed discussions about end users and end use, and the workshop team decided jointly on the areas to be investigated further and on the areas to be addressed for maximum benefit.

As a result of this workshop it was decided to do the following three things:

1 To develop a Pricing Workshop for people from Countries and Business Divisions who were concerned with a given pricing situation. They would use their live situation as the core of the training, with the workshop providing a controlled environment for a free exchange of information.
2 To provide some kind of computer aid to support their thinking process, to speed up the pricing calculations and to keep a record of their thoughts. This tool would be introduced into the Pricing Workshop.
3 To improve the regular exchange of information between Countries and Business Divisions.

The rest of the paper concentrates on item 2 of this list.

1.3 Determining tools and priorities

Because there was only a small amount of funding immediately available it was decided to develop a prototype to test ideas and to provide a demonstration that could be used to get further funding. REVEAL on VME was chosen as the prototyping tool because it was known to be fast, VME time was available to MD&T and this would give some form of central control during trials. It would also allow Workshops to be run in hotels around the world, using a portable OPD connected to the VME machine at Beaumont over the public network.

At about this time another group of mainly business planners and marketeers was discovered, which had good divisional and corporate representation. They were looking at the marketing process overall, seeking ways to improve it. The pricing training team linked up with this group and reviewed the priorities for investment by setting up a joint working group.

This working group met for the first time in October 1987, and was asked to recommend which of the marketing processes and documents should be tackled first. The main processes were:

- segment selection and identification
- description of product requirements
- pricing policy
- promotion
- distribution channel
- Differential Resource Analysis
- impact of market share on profit
- Profit and Loss (P&L)

and the documents to be considered were:

- the Market Requirements Statement: to obtain initial funding for a new product
- the Market Introduction Plan: to define, and gain company commitment to, the introduction and life cycle management of the product
- the Market Introduction Document: to authorise release of the product to nominated countries.

Each of these processes and documents was assessed against a number of factors, including the following:

- was it a mandatory process?
- how difficult was it to create and update?
- what percentage of marketeers had to do this often?
- could it be easily implemented as a system?
- how quickly would benefits appear if it were implemented?
- would it provide data for other parts of the system?
- would its effective use depend on creating a database first?

The conclusion was to continue with pricing and a simple P&L as the starting point, but to plan to gain funding for the whole exercise.

1.4 Introducing the prototype

Work started on the initial prototype in the autumn of 1987 and the first two-day Commercial and Pricing Workshop was held in an hotel in January 1988. The prototype expert system CAPS – Computer Aided Pricing System – was used for the first time: the meaning of the initials has been changed

since then to Computer Aided Planning System, as it now does much more than just pricing. After presentations and discussions of markets, competitors' offerings and prices, CAPS was used to review the ICL prices. From this very first use of a fairly basic model the benefits of improving the quality of the analysis and the speed of obtaining the results were obvious. The system is described in section 4.1 below; this early version covered only parts of items 3, 4, 5, 6 of the CAPS main menu, with very basic help screens. (See Fig 1).

After use in a few more Commercial and Pricing Workshops the expert system increased in scope and began to be used in Applied Marketing Workshops, which cover the whole marketing planning process from segment selection to Profit and Loss. A Marketing Facilitator leads the group of segment and product marketeers and business consultants through the marketing process, leaving them free to concentrate on providing information for the plan.

When these workshops were first run they took four days; with experience in setting up and running them, together with the use of CAPS, this is now reduced to two days.

1.5 Use and benefits

Using CAPS during a workshop allows the participants to provide more detailed information on a marketing plan and gives them the time to sub-segment markets and so to develop more precise strategies. The overall increase in the quality and productivity of the work done is probably factor of two to three times, achieved because:

- CAPS does all the calculations, re-calculations and aggregations. Previously workshops could get bogged down in the calculations for just the introduction phase of a marketing plan, and detailed planning for growth, maturity and decline was rarely done during a workshop. If it was suggested that the plan had assumed an incorrect market adoption cycle there was no possibility of restarting without completely demoralising the group. With CAPS the plan can be revised completely by changing only one or two fields on the Market Size and Market Life Cycle screen
- information entered for one of the four phases of the market life cycle can be copied across to another phase for discussion and development. Similarly whole segment plans can be created by copying an existing plan and modifying it. This is particularly useful when sub-segmenting
- sets of segments can be selected and aggregated at the P&L level, supporting portfolio planning for both segment and product managers
- provided only that the user understands ICL's marketing language CAPS is easy to use, with good HELP facilities: these are described in section 4.3 below
- CAPS leads the user through the ICL marketing process; a route map is provided to remind users where they are in this – see section 4.2. Some

fields must be completed before proceeding, and for key fields the user must declare whether the information is soundly based or just a guess.

To emphasise the reality of these benefits, one group of marketeers has analysed 13 market segments in a series of workshops. As a result they have decided to withdraw from 4 of these, to change the distribution channel to third-party for 4 others and to resource the rest fully: the planned consequential increase in profit is £3m.

In its field trials to date CAPS has been used to help 32 users produce plans for some 80 segments, the most active user having 24 segments. About a third of these plans are in active use, the remainder concerning segments that have been rejected for one reason or another. Benefits achieved by users outside the workshop environment have not yet been surveyed.

1.6 Developing the marketing process

Applying computer aid to a process both scrutinises the logic of the process and allows enhancements to be considered. The following enhancements to the ICL marketing process have been introduced with CAPS:

- recording the marketeer's level of confidence in the information he provides. This can be used now in marketing audits, and potentially to list areas for further market research and to provide an overall confidence level for each plan
- introducing further P&L formats and analyses for different audiences, in particular for third parties in the distribution channel
- allowing for weighting factors in the Differential Resource Analysis (DRA), including default values. DRA is the process of comparing the forecasts of competitors' overall spends in the segment, broken down by Product, Price, Promotion and Place (distribution) – sometimes called the 4 Ps of marketing. This is used to determine market shares. Use of weighting factors allows more emphasis to be placed on one than on another – for example, on Price in a price-sensitive market. Weighting can be applied also within a category – for example, within Promotion by giving more weight to spending on extra sales staff than on advertising
- because of the ability to iterate with ease, planning for a target market share as well as deriving this from DRA.

1.7 Future developments

The CAPS expert system has proved its worth in its field trials and its future development and adoption across the company has been agreed. The next release will contain a number of improvements and enhancements whose need was shown by the trials; these will include:

- easier sign-on
- faster navigation
- more sophisticated phasing of P&L
- interpretation of "efficiency frontier"
- obtaining feedback from users
- guidelines for optimisation in DRA
- options for line-printer listings.

Beyond this there are plans to:

- integrate with Officepower to support fully the creation of the documents that are required
- port to the UNIX range, to provide an alternative to VME
- integrate with the company business planning processes and models.

There are longer term developments under discussion, to link more directly with the external databases of independent service providers, with ICL's own systems such as Product Database and Configurer, and with other ICL workbenches such as the future Designer's Workbench.

2 Designing the Expert System

2.1 What is an Expert System?

An Expert System can be defined in broad terms as a computer system which allows a user with no (or very limited) expertise in the field under consideration to follow some sequence of questions or requests for input, about some problem or situation, and arrive at a conclusion similar to that which would be reached by an "expert" in the field.

The field of medicine has produced some well-known expert systems where a junior or inexperienced doctor can be guided through stages of diagnosis to arrive at the conclusion that would be reached by a senior consultant; the expert system holds within itself the knowledge of the field ("domain of knowledge") possessed by the consultant.

2.2 The Knowledge Engineering situation

"Knowledge Engineering" is the term which has been given to the production of Expert Systems; broadly speaking, it covers the two operations of knowledge elicitation and of building the knowledge into the expert system. Knowledge elicitation is the process by which the Knowledge Engineer extracts from the "Expert" the domain knowledge possessed by the latter. Often, this has never been formally set out, and the elicitation process is as much a voyage of discovery on the expert's part as on that of the knowledge engineer. When a prima-facie version of the knowledge is clear, a prototype expert system is built (using whatever tools appear best for the subject concerned) and is tested against the real life situation. This process then

usually reveals gaps in either or both of the knowledge elicited or the expert's understanding of how he goes about his job; the "knowledge" is revised and the expert system modified until the expert is content that the advice or results given by the Expert System are near enough to what he would advise.

A number of working requirements for the knowledge engineering situation become clear:

1 There must be an "expert" whose brain can be picked, or at the very minimum some documentation of the expertise to be built into the expert system.
2 The tools used to construct the expert system must allow rapid construction of the initial system (what is often called "fast prototyping") and easy subsequent amendment of the system.
3 The system must be able to provide an adequate interface between the computer and the human being using it – i.e. the so-called man–machine interface must permit of easy input and acceptable output.
4 The system must be capable of being used by the "computer illiterate", whose knowledge of how to use a computer need be no more than the logging-in process.

3 The CAPS Expert System

3.1 The choice of REVEAL as a building tool

The decision was made to implement CAPS in ICL REVEAL. REVEAL is a high-level language which had already been used in Knowledge Engineering Business Centre (KEBC) to construct expert systems, a notable example being VCMS, the VME Computer Monitoring System widely used by VME customers and within ICL. REVEAL has a number of features which make it suitable for the purpose:

1 Extensive facilities for getting, storing and manipulating both numeric and textual data;
2 Menu-based user interfaces allowing input of data via screen menus and forms;
3 The use of a high-level command language and an "amend and run" data manipulation language, which together confer fast prototyping ability; and
4 it is available under both VME (on mainframe processors) and UNIX (on DRS300 and Sun workstations) giving users potential access via a wide range of devices.

4 Some basic principles of the CAPS system

4.1 Access and Navigation

The user requires a bare minimum of knowledge of the computer's command language to access CAPS. Thus in the case of the VME version, he needs to

know only how to log on to the host computer. Once accepted, he enters the word CAPS, replies to a request for the name of his personal database, and is presented with a CAPS welcome screen cum notice board which records any recent updates to the system. The next screen is the MAIN MENU which offers him a number of options; this is a screen to which the user will return from time to time. When first displayed, most of the options are blocked off from the user – these option numbers are displayed in brackets ( ) and any attempt to select them will result in a message that the option is not yet available.

```
|_CAPSMAIN                          CAPS                          01/08/89 15:49
                                  MAIN MENU        Database in use: CAPS
                                  =========

        Options:
           1   -   Start a new Segment Marketing Plan
           2   -   Select a previous Segment Mkt Plan
          (3)  -   Segment Definition
          (4)  -   Market Life Cycle
          (5)  -   Perceptual Maps
          (6)  -   Efficiency Frontiers
          (7)  -   Transfer Costs
          (8)  -   Channel & Development Costs
          (9)  -   Market Share by DRA
         (10)  -   P & L and statistics
          11   -   Segment Aggregation
         (12)  -   Save the results of the session
          13   -   Delete segment data
           Q   -   Quit i.e. leave CAPS
               Option numbers in parentheses will not be accepted.

                                  Option:   _? _

        To access a routemap, enter ! on any screen other than this menu.
```

Fig. 1 CAPSMAINMENU as displayed on entry

As the options are successively worked through, the brackets are removed (allowing the user to return to, and amend his input for, a previous option). At the same time a column of asterisks and letters appears at the right hand side of the main menu, showing which options have been completed. Some of the options require data for the four different phases of the Market Life Cycle, Introduction, Growth, Maturity and Decline, and entry for these is indicated not by an asterisk but by the appropriate letter, I, G, M or D. Thus the main menu for a retrieved, partly completed, segment model would appear like Fig. 2.

Note that some options are available at all times – the workspace can be loaded with an empty model "skeleton" (called a template) to allow the creation of a new segment model; or an existing (perhaps only partially

```
|_CAPSMAIN                          CAPS                          01/08/89 14:54
                                  MAIN MENU        Database in use: CAPS
                                  =========

Options:                                          DEMONSTRATION.10 is loaded
   1  -  Start a new Segment Marketing Plan
   2  -  Select a previous Segment Mkt Plan  *
   3  -  Segment Definition                  *
   4  -  Market Life Cycle                   *
   5  -  Perceptual Maps                     IGMD
   6  -  Efficiency Frontiers                IGMD
   7  -  Transfer Costs                      IGMD
   8  -  Channel & Development Costs         IGMD
   9  -  Market Share by DRA                 IGMD
  10  -  P & L and statistics                *
  11  -  Segment Aggregation
  12  -  Save the results of the session
  13  -  Delete segment data
   Q  -  Quit i.e. leave CAPS
      Option numbers in parentheses will not be accepted.

                                  Option:   _? _

   To access a routemap, enter ! on any screen other than this menu.
```

Fig. 2 CAPSMAINMENU partly completed

completed) segment model can be loaded; or an aggregation of a number of segments can be set up and executed; or some housekeeping can be done, such as the deletion of unrequired versions of a model.

Navigation from option to option depends on whether the user is creating a new model from scratch, or whether he is revising or extending an existing model. In the first case he is led from one option into the next automatically; in the second, at the completion of an option he may choose to return to the main menu, or by typing "〉" he may proceed to the logically-next option. In general, he may also review at any point the previous screen in a sequence of screens, by entering "〈". He may also jump from an option to the main menu, by typing CTRL/Q.

Those options that require similar operations for each of the four Market Life Cycle phases start with a phase selection screen (see Fig. 2).
and they finish with a steering screen which allows the data entered for a particular phase to be copied into any of the other phases (as a basis for rapid completion of other phases (see Fig. 3)).

and they finish with a steering screen which allows the data entered for a particular phase to be copied into any of the other phases (as a basis for rapid completion of other phases (see Fig. 4)).

```
|PHASE                          CAPS                    01/08/89 14:57
DEMONSTRATION.10        PERCEPTIONS - MLC PHASE
                        ======================

      Please indicate the MLC phase for which you are going to
      input Perceptions Data:

          Options:   1  -  Introduction

                     2  -  Growth

                     3  -  Maturity

                     4  -  Decline

                     Q  -  Quit i.e. return to main menu

                           ?
```

Fig. 3 PHASEMENU

```
|EFSTEER                        CAPS                  08/01/89 15:06
DEMONSTRATION.10                                    INTRO - 4 qtrs
£K                                                  1988 Q1 - 1988 Q4


      Enter > to proceed to the transfer cost screen
        or  < to return to deal with another phase
        or  I,G,M or D to copy prices data to another phase
        or  Q to return to the main menu

                              >
```

Fig. 4 EFSTEER

4.2 The "WHERE AM I" facility

The CAPS system involves some 45 different screens, seven data entry sequences and four parallel paths. It would not be surprising if the user sometimes forgot just where he was in the process, or the route by which he had arrived at the present screen. To help him, a WHERE AM I route map is provided. On any screen, a "!" may be typed in any position. The route map is then displayed: see Fig. 5.

On the route map, which is essentially a flow diagram of the CAPS process, the present position of the user is indicated by a !, and those options and phases which have been visited during the present session are indicated with asterisks.

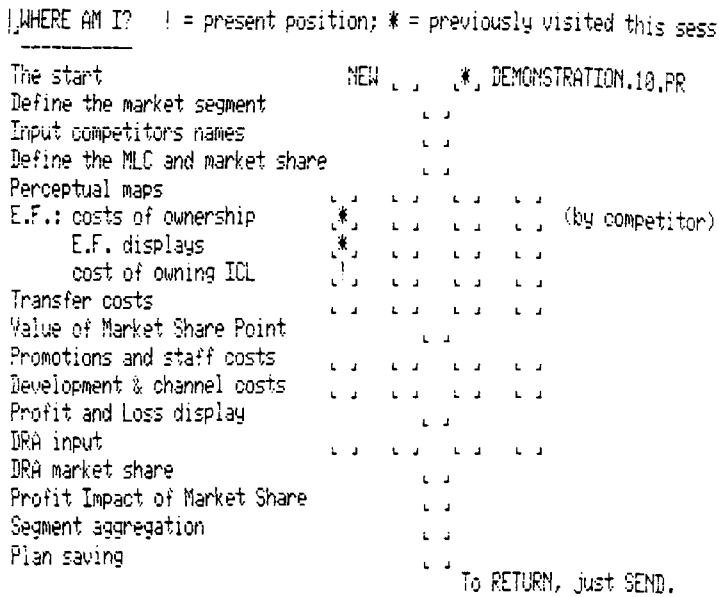


Fig. 5 ROUTEMAP showing a typical sequence

4.3 Two levels of help

Two different help mechanisms are available to the user. The REVEAL menu facility provides an in-built mechanism whereby the input of a "?" on any field on a screen causes a help display for the particular field. An example field help screen is shown: see Fig. 6.

To avoid the use of programming effort in entering and revising field help, a process was devised which will read a host-processor file and insert the help data into the appropriate places in the individual REVEAL menu files. This means that a master file can be maintained on a wordprocessor by the CAPS system authority, who can arrange for the system to be updated without requiring KEBC effort.

During development of the system, the need was identified for an overall help system, to which was given the name "super help". Rather than giving specific help about the data to be entered in a specific field of a screen, it deals with "topics". This is a form of screen-available marketeer's manual, where the "topics" are essentially book chapters, each consisting of a number (up to 9) of pages. Each of the 45 different input screens in CAPS has associated with it a number, of the form topic page. Keying CTRL/H on any screen causes the appropriate topic page (not necessarily the first of the topic) to be displayed. From that page it is then possible to navigate forwards (or backwards) through the pages of the topic, to go to the HELP index, or to return to the screen that was in use. An index screen is shown: see Fig. 7.

```
|_MKTSIZE                                              PD : 16 DEC 88
                        HELP for NUMBER OF BUYERS
-------------------------------------------------------------------------

        This is the total number (N) of purchasers in this Segment,
        over all phases.

        It will be multiplied by the average number of purchases (K)
        to form the total available market (Q) for ICL and its
        competitors.

        As this system allows easy aggregation of segments, the segment
        should be chosen such that N is a viable number, possibly in
        the range 50 - 300.

                                                  Press SEND to return _ _
-------------------------------------------------------------------------
Example: The segment consists of 120 large local authorities that could
each buy between 5 and 10 office automation departmental systems.  We
enter 120 in this field, and set K at around 7.
```

Fig. 6 A FIELD HELP SCREEN FOR MLCMENU

```
|_1.1                      CAPS - GENERAL HELP
-------------------------------------------------------------------------

                          Table of Contents
                          -----------------

1  Table of Contents                    20 Market life Cycle
3  Scope of the CAPS system             22 ICL Target market share
                                        24 Innovators/Opinion leaders
5  Navigation
                                        26 Perceptual Axes
7  Item level HELP                      27 Perceptual Map interpretation
9  General/Context HELP
                                        28 Efficiency Frontier
11 Starting a new Segment               29 - Competitor price buildup
13 Retrieving a previous Seg            30 - Frontier interpretation
16 Segment definition                   31 - ICL offer description
18 Competitors


                                                        more....
-------------------------------------------------------------------------
Return R, Index I, Topic nn or nn.n, Back B, Continue _C _
```

Fig. 7 A SUPERHELP INDEX SCREEN

Again, the principle that no programmer effort should be needed to maintain or update the super help system is applied. The CAPS system authority can maintain two wordprocessor files, one the pages of the help text and the other a list of CAPS screen names and associated topic numbers. Two utility programs can then be invoked to update the CAPS system with this data.

4.4 On-screen display of computed results

Most of the screens presented to the user require numerical input, on which some kind of computation takes place. On a screen which is computationally self-contained, it is general CAPS practice to display the result of the computation on the screen and redisplay it to the user. He can then confirm that his input is satisfactory, or can correct any error that the result has shown.

An example of this kind is the "cost of ownership" screen, in which the user enters a number of component costs, and possibly a discount percentage, and sub-totals and totals are computed and displayed to him.

```
OWNERSHIP                          CAPS                          02/08/89 13:54
DEMONSTRATION.10          COMPETITORS PRICES AND TCOO            GROWTH - 4 qtrs
£K                        ==========================             1989 Q1 - 1989 Q4
COMPETITOR 1 : MAXCOMM
Description: PROJECT Q                    Usage Life 4 years
                          Price per Periods  Value   Source ref.    Cred.
Equipment S.P.            10       1          10.0   A                1
Software OTLC             3        1           3.0   B                1
Software ongoing          .25      16          4.0   C                1
Ongoing services - Equipt 1.5      4           6.0   D                1
        - Software support         1
Professional services     3        1           3.0   E                1
Training                           1                                   
Other supplied products            1                                   
System discount 10      %                      F                    1
  SUBTOTAL - TCOP                             23.4
Cust.o'heads - space               1                                   
  - power and facilities           1                                   
  - communications                 1                                   
  - staff                 4        2           8.0   F                1
  - other                          1                                   
TOTAL COST OF OWNERSHIP                       31.4
To return to previous screen enter < and SEND, or else SEND when complete
```

Fig. 8 OWNERSHIPMENU

A rather different kind of results display is used to display perceptual maps and efficiency frontiers, marketing concepts which are based on estimates of competitor performance on two parameters considered to be important to the would-be buyer. Here the implementation problem is to construct an

adequate graphical display given the limitations of character terminals – the following screens show examples of a perceptual map and an efficiency frontier which are in practice found to be perfectly adequate.

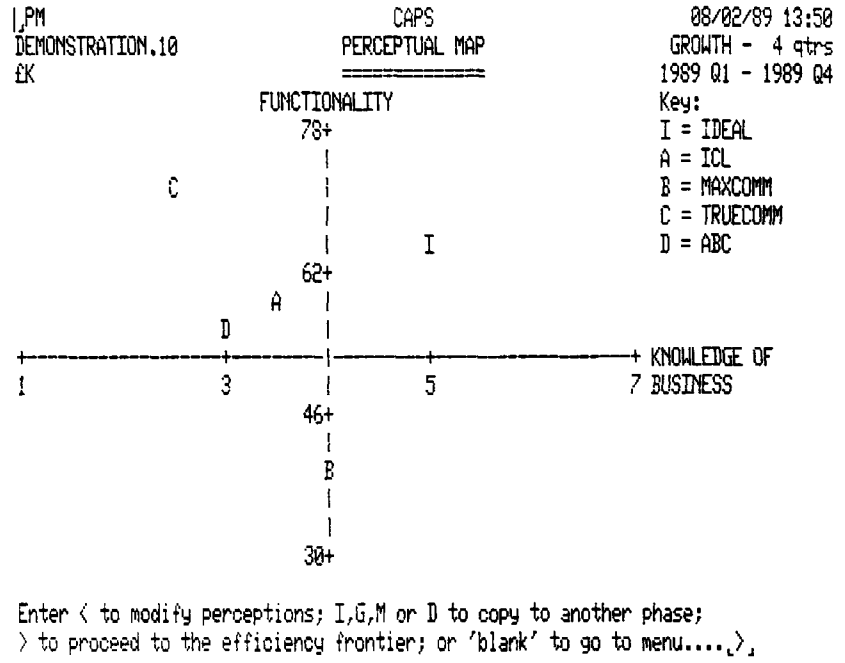


Fig. 9 A VME-TERMINAL PERCEPTUAL MAP DISPLAY

5 Future enhancements

At the time of writing (May 1989) CAPS provides the marketeer with all the facilities implied by the options listed in the main menu shown in section 4.1, which is essentially the mechanisation of the numerical processes associated with the assessment of ICL and competitor comparative performance, and the consequent market share and contribution to the company's Profit & Loss account. This represents only a first phase of a Marketeer's Workbench – an important next stage is the integration of CAPS and intelligent ("expert") word processing, with access to company databases of marketing data, to ease and simplify the production of marketing documents such as the Market Requirements Statement.

Attention is being directed also to other delivery vehicles. Much of the initial use of CAPS has been made with OPDs and a VME service located at ICL's Beaumont education and training centre; the use of more up-to-date local computers and workstations is being investigated. This would give independence from distant mainframes and communications links, except when some

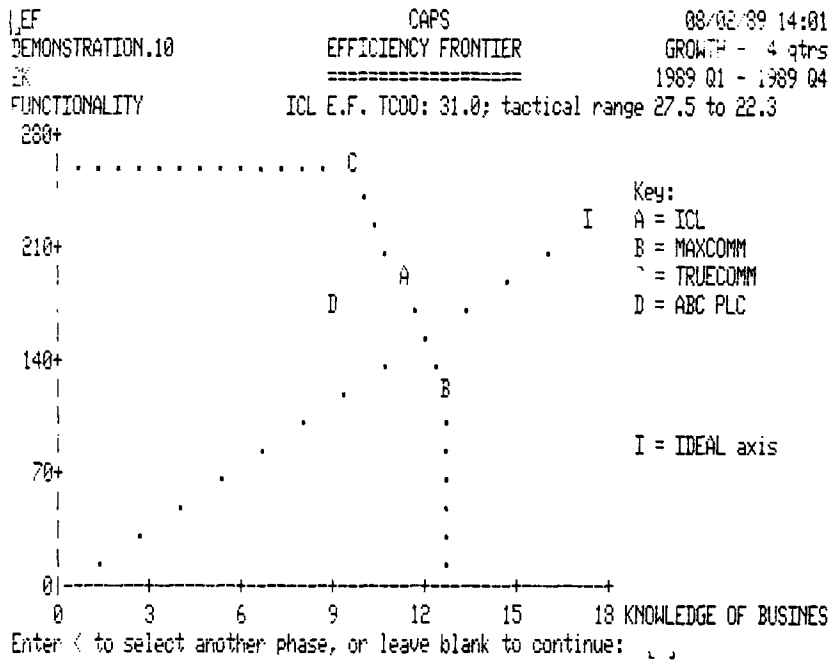


Fig. 10

kind of database access was required, and would allow the generation of much more sophisticated graphical output.

Acknowledgements

Developing CAPS has been a somewhat iterative process, in which an initial trial release was made when only a few facilities had been implemented, with subsequent releases as facilities were amended or added. These partial versions have been used in Applied Marketing workshops over a period of about a year. Our thanks go to:

Colin Simmons, who has acted as facilitator during the workshops, as the principal channel for bug information and has been the "expert" responsible for the mass of "help" data built into the system

John Knott, who now manages the project on behalf of MD&T and whose persuasiveness has guaranteed the funding for the development work

Harvey Dodgson, who provided the seedcorn money for the prototype and all the members of the ICL Marketing Council, who have continued its funding.

The Knowledge Crunching Machine at ECRC: A Joint R&D Project of a High Speed Prolog System

H. Benker, M. Dorochevsky, J. Noyé, B. O'Riordan*, A. Sexton, J.C. Syre
ECRC – European Computer-Industry Research Centre GmbH, Arabellastr. 17 D-8000 München 81, West Germany

Abstract

The KCM (Knowledge Crunching Machine) system provides a fast and user-friendly Prolog environment catering for both development and execution of significant Prolog applications. It consists of the high performance back-end processor KCM, coupled to a UNIX† desk-top workstation. The most salient aspects of its hardware and software are presented here. Some early benchmark results obtained on prototype machines show that KCM compares favourably with other dedicated Prolog machines and available commercial systems running on fast general purpose processors. It runs at a peak speed of 833 Klips (Kilo Logical Inferences per Second) on list concatenation.

1 Introduction

KCM (Knowledge Crunching Machine) is a research project carried out at ECRC, the joint research centre of ICL, Bull, and Siemens in Munich, from 1987 to 1989. The main objective of this project was to develop of a high-performance Prolog processor system.

The KCM system is designed as a single user, single task high-performance back-end processor which, coupled to a UNIX desk-top workstation, provides a fast and user-friendly Prolog environment catering for both development and execution of significant Prolog applications.

KCM is a follower of two former projects in 1985 and 1986, investigating different methods to design and connect a dedicated Prolog engine to a host machine. These projects, called ICM3 and ICM4, led to the design and simulation of two different architectures (a tightly coupled co-processor, and a stand alone, RISC-like system). These designs were implemented on a CAD system, and were simulated at the component level. An appropriate set of

*ICL Bracknell, Future Systems Technology, Lovelace Road, Bracknell RG12 4SN, England.
†UNIX is a registered trademark of AT&T.

software was written to produce microprograms, and executable object code from the Prolog high level language (compiler, assembler, ...). The evaluation results appeared to be promising enough to go ahead with more ambitious objectives. At that time, ICM3 was shown to deliver an average performance one order of magnitude higher than available Prolog software systems (BIM, Quintus, or IF Prolog), and a peak performance of 450 Klips (kilo inferences per second), equivalent or better than all other Prolog hardware systems.

In February 1987, the decision was made by the shareholder companies to pursue the research with an actual implementation of a slightly different system that would connect to each of the shareholder UNIX workstations, and whose performance objectives would be equivalent to ICM3 or higher. The original aim defined KCM as a research project leading to three running prototypes entirely developed at ECRC in 1988, and to the delivery of around 50 pilot systems, manufactured by Siemens, and delivered by each company to selected users of their choice. The project was viewed as a truly integrated project, where Bull, ICL and Siemens would play a significant role during the research and development phases, and a more crucial role in the final steps of manufacturing and delivering the pilot systems.

At ECRC the project was made possible by the close teamwork of more than ten people in the Computer Architecture Group led by Jean-Claude Syre. In addition to the authors themselves the following people have also been working as full time researchers on the KCM project: Jean-Michel Beacco, Sylvie Bescos, Lindsay Errington, Thomas Jeffré, Anita Pöhlmann, Bruno Poterie, Olivier Thibault, and Günter Watzlawik.

The KCM project has continuously received assistance and support from the Logic Programming Group of ECRC. Special thanks are due to the team in charge of the Sepia Prolog system, and particularly Micha Meier for his time and expertise in all software aspects of advanced Prolog implementations.

Outside ECRC, numerous people in all three shareholder companies have contributed to the success of the project. There has been joint development work with Bull/GIPSI, Siemens and ICL for the KCM interface boards. The memory boards were jointly designed by ECRC and Siemens, and were manufactured at Siemens. The host-specific software was developed by Bull/GIPSI, and tuned by people of each company for their particular workstation. At Siemens in Munich, a team of four people, led by Walter Woborschil and Peter Stock, has been working since October 1988, to prepare the integration of the KCM machines at the factory in Augsburg.

We acknowledge the invaluable support of Bill O'Riordan of ICL, Chairman of the KCM Project Board, created in early 1988 to manage and control the development of the project and the relationships among the companies. The KCM Board has provided the technical team at ECRC with important technical and managerial decisions and has played a key role in the final phases. The KCM Project could not have reached its current state without

the strong backing of Peter Mueller-Stoy, head of the SYS department at Siemens Perlach, and Owen Evans of ICL Bracknell. Thanks are due too to the people in Bull who gave their support to the project: François Sallé, François Anceau, and Peter Chan.

So far four prototype machines have been built. Operational since July 1988, they have been tested and used at ECRC and Siemens. At the time of writing of this paper (mid 1989) a full software environment is nearing completion. At the Prolog level, this environment KCM-Sepia is based on Sepia [Meier89 1989], a second generation Prolog system also developed at ECRC.

The shareholder companies will deliver 50 KCM pilot systems, manufactured and integrated by Siemens, at the end of 1989. Moreover, a Support Group has been set up to complete the KCM systems with the appropriate documentation, to provide their hardware and software maintenance, and to start possible further developments. It is located at ECRC, and staffed with people from all three shareholder companies. Headed by Thomas Jeffré, the Support Group comprises Hans Johnen, Stefano Novello, Jonathan Price, Karl-Heinz Seidel, and more people to come in 1989.

The paper is organized into several sections. Section 2 introduces the main features of the KCM system architecture. Section 3 explores the different hardware components of the KCM engine, and section 4 describes the software architecture, and its different elements. Section 5 presents evaluation results, and section 6 gives some conclusions about the KCM project, a truly collaborative work between Bull, ICL, Siemens, and their research centre, ECRC, and also an illustration of a fruitful cooperation among people of different European countries for their mutual interest and benefit.

2 System architecture

2.1 A back-end processor

KCM is a back-end processor connected to a UNIX host rather than a stand-alone workstation like PSI [Nakashima and Nakajima 1987] or a coprocessor like X-1 [Dobry 1987].

The stand-alone approach was discarded because it requires the development of much peripheral hardware and the associated software. Also, in order to use existing applications (e.g. databases, window manager) it is necessary to port a standard operating system with its complete environment to the machine.

In the co-processor approach, sharing the data between the host and the co-processor requires identical data formats on both processors. This would be a non-optimal architectural compromise for at least one of the processors. It also means sharing the memory bandwidth between the two processors,

which we considered to be unacceptable as far as the performance requirements were concerned.

As a consequence KCM is equipped with its private memory and runs its own operating system in an environment as shown in Fig. 2.1. The cost to be paid for the loose coupling to the host are some limitations on the granularity of host-KCM interactions.

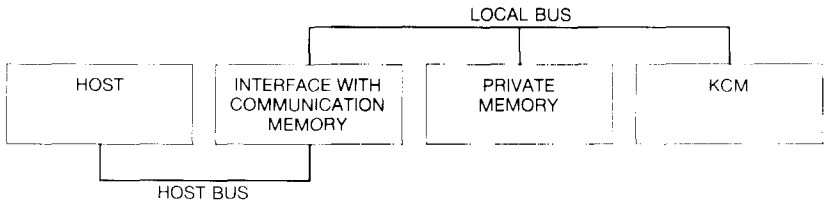


Fig. 2.1 The KCM system

So far, the hosts considered for the pilot systems are: ICL's PWS (Intel 386 based), Siemens' MX-300 (NS 32032) and BULL's DPX-1000 (M68020), also host of the prototype systems.

2.2 Performance, flexibility and functionality

A straightforward way to provide performance is to hardwire the "WAM", an abstract instruction set defined by D.H.D. Warren that has become the most widely accepted implementation technology for Prolog [Warren 1983]. This has been the approach taken by PLM [Dobry et al. 1985] and X-1 [Dobry 1987]. This drastically limits the functionality and flexibility of the machine. For instance, it provides neither basic facilities like simple arithmetic nor the mechanisms necessary to implement a complete Prolog runtime.

This has been overcome in PSI by implementing a number of basic built-in predicates in microcode. However good the microprogramming environment is, this is bound to raise maintenance problems and is difficult to upgrade.

Much care has been taken to keep the design of KCM as flexible as possible. High-level instructions, the *Prolog Instruction Set*, implement the basic Prolog mechanisms (unification, backtracking, indexing ...). These instructions define an enhanced version of the WAM, adapted for implementation in hardware. Low level RISC-like instructions constitute a general purpose instruction set, the *Basic Instruction Set*, used to implement the functionality that the standard WAM does not provide. These instructions are in no way tied to Prolog and make it possible to efficiently implement on KCM other languages, e.g. Lisp or even C. To facilitate such implementations, the kernel of KCM (providing I/O and memory management functions) and the low-level code generation tools are language independent. As a result, though

KCM is dedicated to Prolog, it is not restricted to Prolog and can be seen as a tagged general purpose machine with support for logic programming.

KCM has been designed not only to execute significant applications, but also to take advantage of its high performance for development purposes. KCM-Sepia provides the standard Prolog development facilities, i.e. an incremental compiler and a powerful debugger.

It also includes advanced Sepia features, such as a sophisticated module system, coroutining and event handling.

3 Hardware architecture of KCM

3.1 Technology

Physically the KCM system consists of three boards: the processor board, the interface board and the memory board. All three together fit in a workstation cabinet. The processor board has a size of 30 × 50 cm, whereas the other boards are standard double-Europe format.

For the implementation of the KCM processor board we chose to use off-the-shelf technology. Most of the glue logic is implemented in 74AS and 74F series of chips and fast PALs (Programmable Array Logic). The core of the CPU is built around the AMD 29300 series of 32-bit components. In addition two 1·5 μm CMOS ASIC chips are used to comply with the restrictions in board space.

SMD technology with components mounted on both sides is used for the memory board. One such board holds 32 MBytes, parity and the on-board RAM controller. This is currently implemented with 1 Mbit chips, but the layout of the board is compatible with the use of 4 Mbit chips to obtain 128 MBytes. Up to two such boards can be used with KCM.

3.2 Block diagram of KCM

The hardware architecture of KCM is based on the Harvard architecture, i.e. it has two separate access paths to memory, one for code and one for data. There are two independent caches, but physical memory is shared between code and data. KCM shares this property with most RISC architectures as well as some of the recent CISC designs (e.g. Motorola 88000 and 68030).

KCM is an entirely synchronous machine, controlled by a single central control unit with a cycle time of 80 ns. Its microcoded operation allows rapid change of control flow which is essential for an efficient implementation of logic programming.

Operated by the control unit, the execution unit and the prefetch unit perform the actual execution of the programs. Each of those units is

connected to its own cache. The prefetch unit is closely linked to the code cache from which it prefetches instructions. Those instructions are then executed in the execution unit, which accesses data in the data cache.

Both prefetch and execution units are directly connected to their respective caches, i.e. there is no address translation involved in accessing the caches. The memory management unit translates the virtual addresses to physical addresses only when physical memory is accessed on a cache miss. Nevertheless it verifies the access rights, based on virtual addresses, for every access to the data cache.

The block diagram of KCM in Fig. 3.1 shows how the different units are connected together. All major buses in the machine are 64 bits wide to transfer a complete word, including tag and value in a single cycle.

The WAM model of computation supposes tagged data words, i.e. a basic entity consists of a value plus an additional tag field that gives information on its type. As in SEPIA a length of 32 bits for each tag and value was chosen, leading to a word length of 64 bits.

Now we can summarise the major characteristics of KCM:

- 64-bit tagged architecture
- conventional technology (TTL/CMOS) plus two CMOS-ASICs
- microcoded control
- 80 ns cycle time
- separate logical code and data caches, 8K words each
- private memory (32 Mbytes on one board)
- hardware support for the basic Prolog mechanisms (indexing, unification, backtracking)
- verification of access rights to virtual addresses
- 4 Mbytes of dual-ported memory for communication with the host (on the interface board).

3.3 The CPU

3.3.1 Basic data manipulation: Source and destination of all data manipulation instructions are registers in the 64 × 64 bit register file. As with RISC processors there no instructions to directly modify a memory location.

The instruction format of KCM allows specifying four addresses: two source and two destination registers. Figure 3.2 shows how this can be used to perform two register moves in one cycle: data are output on the buses ABUS and BBUS. They are transferred via the ALUs ALU_C and ALU_D and written back to the register file on the buses CBUS and DBUS respectively.

For arithmetic and other data manipulation the ALU_D, the FPU, or the TVM are used: the ALU_D supports 32-bit integer arithmetic; the FPU

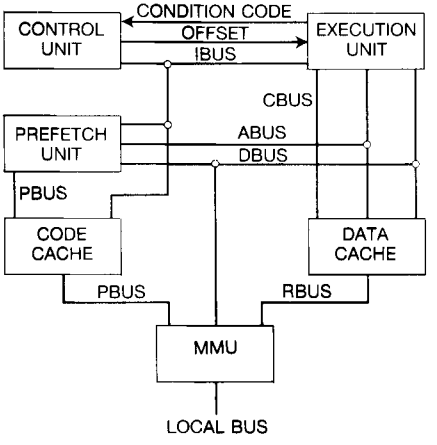


Fig. 3.1 KCM top level architecture

implements floating point arithmetic on 32-bit IEEE format; and the Tag Value Multiplexer TVM is a unit to do basic operations on 64-bit words and thus make the tag part of a word accessible.

3.3.2 Memory access: KCM supports three addressing modes: direct, pre-address calculation and post-address calculation. All addressing modes allow address computation in a single cycle. As on a RISC processor no complex addressing modes that require microcoded calculation of the address are supplied.

The direct addressing mode uses an instruction format that supplies an absolute address of a memory location and a destination or source register for a load or store operation respectively.

The pre- and post-address calculation instructions use an instruction format with three register numbers and a signed 16-bit offset. The registers are:

- Ras – address source register
- Rad – address destination register
- Rds – data source register for store instructions
- Rdd – data destination register for load instructions.

The 16-bit offset is added to the contents of Ras and the result stored in Rad. The cache is either addressed with the original address in Ras (pre-address calculation) or the address that gets written into Rds (post-address calculation).

These addressing modes allow implementation of stacks that grow in either direction. Pre-increment, post-increment, pre-decrement, post-decrement, as well as an offset from the top of the stack are all supported.

3.3.3 Support for Prolog: An important feature of logic programming is unification, a kind of two-way pattern matching. To efficiently support unification it is essential to rapidly test the types (tags) of two objects stored in the register file and to take appropriate action. In KCM this is implemented using a 16-way branch facility in the microcode.

One of the characteristics of unification is to often create chains of pointers. KCM allows following such chains of references (dereferencing) at the rate of one pointer per cycle.

Prolog execution relies heavily on backtracking. This is a mechanism not known in conventional programming languages that allows coming back to the state of computation at a certain point and trying alternative solutions if the preceeding attempts failed. The high memory bandwidth of KCM allows saving and restoring of the state of computation in a minimum number of cycles.

KCM has special hardware support to determine at what point of the computation it is unavoidable to save the state. Using these techniques known as *shallow backtracking* and *delayed choice point creation* much of the saving and restoring overhead is avoided.

For more details refer to [KCM 1989].

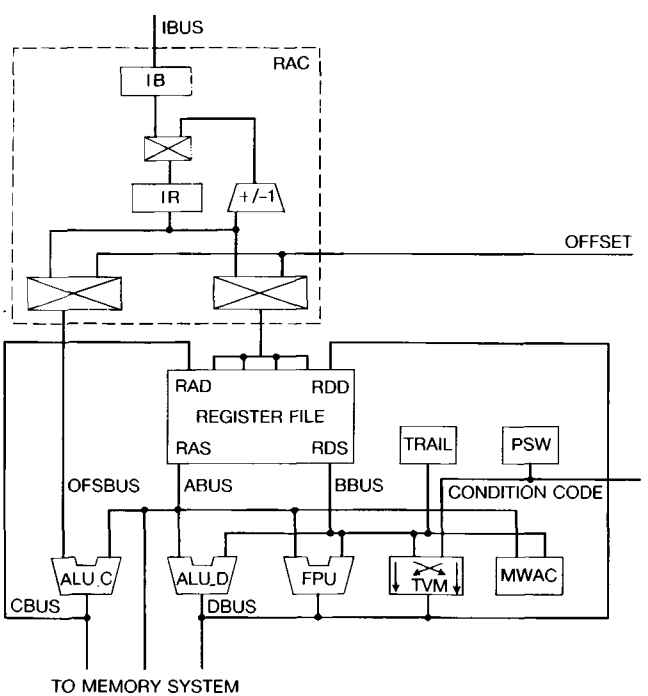


Fig. 3.2 The execution unit

3.4.1 The caches of KCM: Each of the two caches in KCM has a size of 8K × 64 bits. The line size in both caches is one.

Prolog shows a much higher rate of writes to memory than conventional programming languages. Therefore it is important that the data cache be a store-in cache which only accesses memory when a cache miss occurs. On the other hand, almost all accesses to code are read operations and therefore it is affordable to design it as write-through cache, i.e. each write to the cache will also cause a write to main memory.

The standard WAM uses three stacks and KCM-Sepia uses up to seven different stacks. Simulations showed that for accesses to data an associative cache does not much improve performance as long as it is ensured that two stacks do not compete for the same cache locations at any one time. As a result the data cache of KCM is split into eight sections of 1K × 64 bits. Each of these sections is allocated to a particular stack and therefore collisions between different stacks are avoided.

3.4.2 Check of access rights at the logical level: Both caches are directly connected to the CPU and are accessed using virtual addresses. The address translation mechanism of the MMU is not accessed in that case, which prevents it from using its page tables to determine whether an access is valid or not. However, the MMU has the capability to watch the accesses to the data cache and check their validity without access to the normal page tables.

This security mechanism protects the machine against stack overflows and a number of software errors. Amongst other things it also verifies the type of the address. If, for example, an attempt is made to use a floating point number as an address, a trap will be generated.

4 Software architecture of KCM

The basic software architecture design of the KCM system was driven by the goal to provide a fully interactive programming environment suitable both for development and execution of large Prolog applications. This requires:

- on KCM: a complete Prolog system
- on KCM and the host: software to connect the Prolog system to the outside world (keyboard and screen, file system, secondary storage, etc.).

An implementation of the Sepia (Meier89 1989] Prolog system, called KCM-Sepia, runs on KCM. Sepia (Standard ECRC Prolog Integrating Advanced Applications) has been developed at ECRC within the *Logic Programming Group* for standard UNIX based workstations. Users of KCM-Sepia work with a system that is fully equivalent to Sepia but gain a significant Prolog performance speed-up.

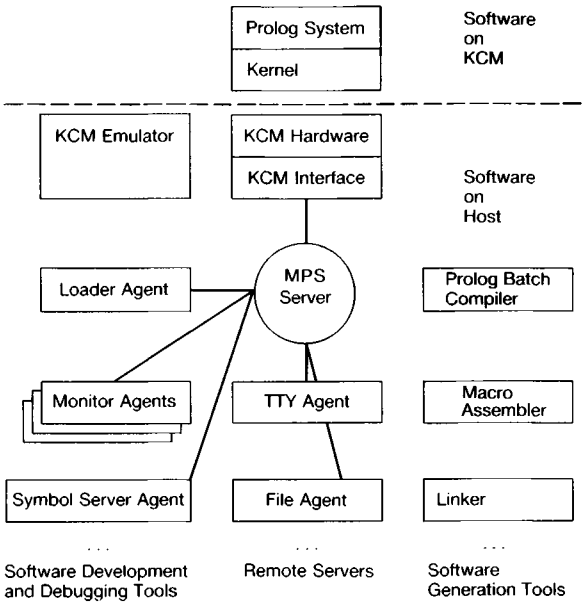


Fig. 4.1 KCM software architecture

Figure 4.1 shows the different components of the KCM software.

The KCM-Sepia Prolog system constitutes the upper layer of the software on KCM. The whole Prolog system resides on KCM and no interaction with the host occurs during Prolog execution except for OS operations such as I/O, paging, etc. The lower software layer on KCM, the kernel, carries out OS tasks, essentially memory management including paging, communication from and to the host, file and terminal access. The Prolog system is completely separated from the kernel and uses system calls to access kernel resources.

The central software component on the host is the message passing system (MPS). A set of separate processes running on the host communicate to each other and to KCM or provide services to KCM. These processes, called agents, connect to the MPS and communicate to KCM either via messages or by DMA under control of the MPS.

One of these agents, the KCM emulator, allowed validation of the abstract machine and testing of major parts of the system long before the hardware was available. It is still used to gather execution profile statistics for evaluation and as a replacement for the KCM hardware.

Software development and debugging is made possible by a loader and several monitor agents which are used to down-load a single program or the

kernel and to debug the system at different levels: microcode, macrocode or Prolog level. Symbolic debugging is supported by a symbol server agent.

The fundamental remote servers for KCM are the file and tty agents. They handle remote file and terminal I/O operations.

Besides the software connected to the MPS, a whole chain of software generation tools has been developed on the host system: a Prolog batch compiler, a preprocessor, a sophisticated macroassembler, a linker and other debugging tools.

4.1 Prolog system

Sepia is a second generation, fully incremental Prolog system. New Prolog procedures and already compiled ones can be dynamically added or replaced using the incremental Prolog compiler. Sepia compares favourably in performance to other commercial Prolog systems and exceeds them in functionality and capability. On top of a classical Prolog system it offers some particularly interesting features:

- a sophisticated module system
- delayed goal execution (coroutining)
- asynchronous and synchronous event handling.

Delayed goal execution may be used to significantly increase the performance in generate and test problems and can preserve completeness by avoiding infinite loops. Asynchronous and synchronous event handling is provided with the goal to support real-time applications. Events may be signals (interrupts) or errors (incorrect argument of a built-in predicate). On top of the asynchronous event handling of Sepia a powerful windowing interface has already been integrated. Sepia's procedure based module system allows structuring of large applications, supports privacy and information hiding but remains transparent to non-modular applications when put in a unique module.

Functional differences between Sepia and the KCM-Sepia are limited to cases where the back-end processor architecture forbids a direct mapping of the Sepia functionality. Tight connections to C functions on the host or to other processes via pipes cannot be provided for free as on a single UNIX machine. A remote procedure call mechanism will nevertheless allow loose coupling to server agents on the host.

However, there are differences in implementation between Sepia and KCM-Sepia. The logical structure has been kept basically the same, but parts of Sepia, which for efficiency reasons are implemented in C, have been directly written in Prolog for KCM-Sepia. This approach costs less development effort without affecting performance given that efficient Prolog execution is guaranteed on KCM. Typical examples are the incremental Prolog compiler and the Prolog reader and writer.

A clear separation between the Prolog and runtime layer is maintained in the KCM-Sepia Prolog design. At system load time the Prolog layer is loaded dynamically by the runtime layer which gives full flexibility for incrementally extending the Prolog system.

4.1.1 Prolog layer: The Prolog layer consists of an incremental compiler, a debugger and about 300 built-in predicates. The incremental Prolog compiler running on KCM corresponds basically to the batch compiler residing on the host. It is a procedure compiler with the following features: indexing on up to 2 arguments, source to source transformation of meta predicates like *if-then-else, or* and *negation by failure* and inline compilation of simple arithmetic or type testing predicates. The debugger is an extended 4 port debugger which fully handles delayed goals. Most of the built-in predicates are implemented in Prolog, only a small number had to be hand-coded in KCML (KCM assembler).

Once the kernel is running on KCM, bootstrapping of the Prolog system requires the batch compiler to be present on the host and the incremental Prolog loader in the runtime layer. Prolog compiled on the host is assembled to a binary object file. The kernel, responding to a keyboard request, loads the essential core of the Prolog runtime system. The Prolog loader, part of the Prolog runtime, loads the generated binary object files and inserts the Prolog code on a procedure by procedure basis into the runtime structures. Loaded Prolog procedures behave the same as incrementally compiled ones. It is therefore possible to keep parts of the Prolog system or additional Prolog packages under binary object form on the host. This has the advantage that the system does not need to be recompiled each time it is booted and that the Prolog sources can be kept away from the end user.

4.1.2 Runtime layer: The runtime layer is the core of the Prolog system. It maintains the Prolog runtime structures which are the dictionary, the library and the property lists and provides the incremental Prolog loader. The dictionary contains the atom table and the library the procedure table and all module information. The Prolog term database and the global variables and arrays predicates are based on the property lists.

The Prolog runtime is implemented in KCML using a powerful preprocessor which allows use of basic control structures of procedural languages like *if-then-else* and *while-do* constructs.

4.2 System software

4.2.1 KCM kernel: The presence of the RISC-like instructions of the *Basic* instruction set have made possible the implementation of a conventional style operating system that provides features such as system calls, I/O services, signal/exception delivery, debugging facilities, memory management, paging, etc. Care has been taken in designing the operating system to

allow the implementation of different language systems such as *Lisp* or *C* as well as *Prolog*.

Although the kernel does not provide multiprocessing facilities to user level programs, the kernel multiprocesses itself in the form of a collection of lightweight processes in order to avoid useless waiting until I/O requests can be completed.

KCM has no peripherals of its own beyond access to a real time clock and 4 Megabytes of dual ported memory, the other side of which is accessible from a host UNIX workstation. Thus KCM acts as a disk-less, terminal-less machine with I/O operations supported through a remote file system and a remote terminal system.

4.2.2 Host software: The UNIX host kernel contains a small driver to support delivery of interrupts from KCM to a user process on the host and mapping of the dual ported interface memory into the address space of user level processes on the host. Aside from this driver, all host code for dealing with KCM operates in user level in order to facilitate the connection of KCM to different host machines.

A message passing system on the host is provided to allow a number of processes to communicate with each other. Agents can be dynamically added and removed from the system and message transfer can have blocking or non-blocking semantics as and when required. This message passing system is broadcast based with message filtering to reduce overheads. The fact that it is broadcast based allows connection of a monitoring agent at any time that monitors all, or some subset of, the message traffic to assist in debugging and development. Messages consist of a 48 byte fixed length header and an optional variable length body with a maximum length of approximately 4 Kbytes.

4.2.3 Communication between KCM and the host: There are two modes for communication between KCM and the host software. The first mode is used for initial booting of the operating system on KCM and for testing and debugging of KCM at the microcode and hardware levels. In this mode it is presumed that there is no kernel running on KCM. An agent wishing to communicate (such as a monitor or a loader for stand alone binary files) broadcasts their appropriate requests that are accepted and handled by a hardware access manager agent (HAM). HAM knows how to do low level operations such as starting KCM's clock, loading microcode, stepping through macrocode and accessing internal registers and memory.

In the second mode HAM is no longer necessary but instead requests are handled by a kernel control agent (KCA). This agent knows only a protocol for forwarding messages from KCM into the message passing system and vice versa. This is the only agent that handles host interrupts generated by KCM and that causes interrupts on KCM from the host. If an agent such as

the remote terminal handler or the remote file handler has data to be delivered to KCM then the kernel on KCM will send a message to the agent, via KCA, telling it to begin a DMA operation and giving it a physical address at which to begin transfer to. While waiting for the operation to complete, the KCM kernel can continue with other work (letting the Prolog system continue if possible, if not then calculating and starting read ahead operations or memory reorganizations, etc.). When the remote agent has finished the operation (successfully or unsuccessfully) then it sends a message to KCM giving the completion status and the KCM kernel can continue.

KCA and HAM together form the KCM interface software referred to in Fig. 4.1.

5 Evaluation

Table 5.1 compares the peak performance of a number of major Prolog machines, CHI-II [Habata et al. 1987], DLM-1 [Pudner 1987], IPP [Kurosawa88 1988], AIP [Kawakita et al. 1988], KCM, PSI-II [Nakashima and Nakajima 1987] and X-1 [Dobry 1987]. The first figure gives the performance of the machines on list concatenation, the second on the naive reversal of a list. Both figures are standard figures used to assess the peak performance of a Prolog machine.

Table 5.1 Comparison with other dedicated Prolog machines

Machine	By	Klips	Word	Comment
CHI-II	NEC C&C	490–?	40	Back-end – multi-processing
DLM-1	BAe	800?–?	38	Back-end – physical memory
IPP	Hitachi	1360–1197	32	Integrated in super-mini (ECL)
AIP	Toshiba	?–620	32	Back-end
KCM	ECRC	833–760	64	Back-end
PSI-II	ICOT	400–320	40	Stand-alone – multi-processing
X-1	Xenologic	400?	32	SUN co-processor

Except for IPP, which is implemented in ECL (20 ns cycle time), DLM-1 is the only machine which is claimed to reach the same level of performance as KCM. However, DLM-1 is, to our knowledge, far from providing the flexibility and the functionality of the KCM software environment.

Of course, KCM provides much more performance than high-end workstations. Table 5.2* compares the results of KCM and Quintus 2.0, one of the best commercial systems, running on a SUN3/280 workstation (M68020 25 MHz, FPU 20 MHz, 16 Mbytes of main memory). On this standard benchmark suite, initially used to assess the performance of PLM [Dobry et

*The holes in the table correspond to programmes which were too small to get significant results.

al. 1985], the ancestor of X-1 developed at U.C. Berkeley, KCM is, on average, almost eight times faster. Further details were reported in a previous paper [KCM 1989].

Table 5.2 Comparison with QUINTUS|SUN

Benchmark		QUINTUS		KCM		Q/KCM
Program	Inferences	ms	Klips	ms	Klips	ms/ms
con1*	4			0·006	666	
con6*	12			0·046	261	
divide10*	20			0·090	222	
hanoi*	767	11·600	66	1·264	607	9·18
log10*	12			0·039	308	
mutest*	1365	41·500	33	4·644	294	8·94
nrev1*	497	3·300	151	0·649	766	5·08
ops8*	18			0·058	310	
palin25*	323	9·330	35	1·220	265	7·65
pri2*	1233	30·500	40	5·239	235	5·82
qs4*	610	11·000	55	1·315	464	8·37
queens*	657	9·010	73	1·182	556	7·62
query*	2888	128·170	23	12·605	229	10·17
times10*	20			0·081	247	
average						7·85

It is clear, that this gap is bound to be lowered (but not filled) with the new generation of very fast RISC chips. For instance, Applied Logic Systems (renowned for its very fast implementations of Prolog, using native code compilers) claim that they will reach 500 Klips on the 25 MHz Motorola 88000. In terms of average performance, we assess that KCM will still be at least three times as fast, although it was built using technology that is already outdated. Interestingly, both machines use a Harvard architecture and provide about the same memory bandwidth. Obviously, the micro-parallelism and the tagged architecture of KCM make better use of the available bandwidth.

Note also that there is still room for improvement by using faster chips, especially on the critical data paths. Using such techniques, one of the prototypes can run with a 60 ns cycle, leading to more than 1 MLips peak performance. We assess that, with only slight architectural changes, this figure could be easily multiplied by 3 by using up-to-date integration techniques. This figure stems from a fine-grained evaluation of the architecture, currently in progress, which aims at precisely determining the improvement brought to the overall performance of the machine by its different units and features.

It is still too early to comment on the behaviour of the system when developing programs or executing highly interactive programs. However, considering the effectiveness of currently available networked file or graphic

servers and the attention brought to KCM-Host communications, good speed-up are expected in these cases too.

As for developing the system itself, the availability of the message passing system has made facilities possible that would not have been feasible otherwise: the whole connection to KCM via HAM and KCA can be replaced by an emulator agent to assist in debugging. Although the emulator only emulates the machine down to the macrocode level and not to microcode it has still provide a workable environment for development and debugging when there were not enough hardware machines available.

6 Conclusion

KCM has effectively demonstrated that appropriate hardware support of basic logic programming mechanisms could break the 1 MLips barrier, revealing the limitations of current RISC architectures in very high-level language support.

On top of this hardware platform, a complete Prolog environment has been implemented, in order to demonstrate and evaluate the capabilities of the KCM Hardware Engine used as a loosely coupled Prolog coprocessor to a UNIX host.

As it stands, KCM provides a high performance, high functionality environment to develop and execute time and space consuming Prolog applications.

It is also a system open for improvement and further research (e.g. X-Windows server on the host, implementation of constraints, implementation of other languages such as Lisp or C, etc.).

Last, but not least, KCM has been (and will still be with the pilot machines release and support) an exciting joint effort of ECRC and its three European industrial shareholders, ICL, Bull and Siemens. A small but important advance in European collaboration.
BOXT

References

DOBRY, TEP: 'A Coprocessor for AI; LISP, Prolog and Data Bases. In: Proceedings of Spring Compcon '87, pages 396–402. IEEE Computer Society, February 1987.

DOBRY, T.P., DESPAIN, A.M. and PATT, Y.N.: 'Performance Studies of a Prolog Machine Architecture'. In: The 12th Annual International Symposium on Computer Architecture, pages 180–190. IEEE/ACM, June 1985.

HABATA, S., NAKAZAKI, R., KONAGAYA, A., ATARASHI, A. and UMEMURA, M.: 'Co-Operative High Performance Sequential Inference Machine: CHI'. In: Proceedings of ICCD'87, New York. 1987.

KAWAKITA, S., SAITO, M., HOSHINO, Y., BANDAI, Y. and KOBAYASHI, Y.: 'An Integrated AI Environment for Industrial Expert Systems'. In: International Workshop on AI for Industrial Applications 1988, pages 258–263. IEEE Computer Society, 1988.

BENKER, H., BEACCO, J.M., BESCOS, S., DOROCHEVSKY, M., JEFFRÉ, TH., PÖHL-

MANN, A., NOYÉ, J., POTERIE, B., SEXTON, A., SYRE, J.C., THIBAULT, O. and WATZLAWIK, G.: 'KCM: A Knowledge Crunching Machine'. In: IEEE (editor), The 16th Annual International Symposium on Computer Architecture – ISCA'89, pages 186–194. June 1989.

KUROSAWA, K., YAMAGUCHI, S., ABE, S. and BANDOH, T.: 'Instruction Architecture for a High Performance Integrated Prolog Processor IPP'. In: A. Kowalski and A. Bowen (editor), Proceedings of the 5th International Conference & Symposium on Logic Programming, pages 1507–1530. Hitachi Research Lab., August 1988.

MEIER, MICHA, AGGOUN, ABDERRAHMANE, CHAN, DAVID, DUFRESNE, PIERRE, ENDERS, REINHARD, HENRY DE VILLENEUVE, DOMINIQUE, HEROLD, ALEXANDER, KAY, PHILIP, PEREZ, BRUNO, VAN ROSSUM, EMMANUEL and SCHIMPF, JOACHIM: 'Sepia – An Extensible Prolog System'. In: (editor), Proceedings of IFIP Congress 89, pages . IFIP, August 1989.

NAKASHIMA, HIROSHI and NAKAJIMA, KATSUTO: 'Hardware architecture of the sequential inference machine: PSI-II'. In: Proceedings – 1987 Symposium on Logic Programming, pages 104–113. IEEE Computer Society, September 1987.

PUDNER, A.: 'DLM – A Powerful AI Computer For Embedded Expert Systems'. In: R.P. van de Riet (editor), Frontiers in Computing, pages 187–201. December 1987.

WARREN, DAVID H.D.: 'An abstract prolog instruction set'. Technical Report tn309, SRI, October 1983.

'FLAGSHIP' PROJECT

Aspects of protection on the Flagship machine: binding, context and environment

S Holdsworth, J.A. Keane, and K.R Mayes
Department of Computer Science, University of Manchester

Abstract

The Flagship machine is a distributed system in which declarative language programs are evaluated using graph reduction. The system software faces two problems: handling state in a declarative world and sharing resources in a distributed world. In order to solve these problems, the Flagship system software makes use of non-declarative features. However, these features are implemented using mechanisms which ensure consistency of shared state and of distributed resources. Aspects of protection are provided by binding and context mechanisms, and there is an explicit protection domain defined by the environment of a computation.

1 Introduction

Flagship is a collaborative research project between the University of Manchester, Imperial College, London, and International Computers Limited (ICL). Its aim is to produce a complete computing system based on a declarative programming style. The project involves three major areas. First, the exploration of the relationship between different declarative language classes and the programming environments for such classes. Second, the design of a machine architecture for the execution of computations in parallel and of low-level computational models that can serve as targets for translations of higher-level language systems. Third, the design of a software environment for the parallel machine. The long term aim of the project is to combine ease and correctness of programming with significant increases in computing performance which can be expected from parallel computer architectures.

The Flagship machine has a distributed physical architecture: a set of processor-store pairs coupled together by a high bandwidth delta network. Programs are evaluated using graph reduction (Peyton Jones, 1987), a technique originally used for evaluating functional programs and which provides a computational model for declarative languages. Each node of a graph is held in a uniquely addressed store location as a packet. Each packet

has a header which contains information about the nature of the packet, and a body which holds items containing addresses of, or *pointers* to, packets, so forming the arcs of the graph. Packet items may also contain base values (Banach et al., 1988). Graph reduction on Flagship occurs as fine-grain transformations of the graph; these transformations are called graph rewrites (Watson et al., 1986). For a description of the evaluation of functional programs using graph rewriting on the Flagship machine, see Watson et al. (1987) and Watson and Watson (1987).

There are two problems associated with such a *distributed declarative system*:

- the resources of each processing element must be shared between programs running on that processing element in a way which is compatible with the declarative approach;
- each program running on the distributed system must be able to use the distributed resources in a way which is efficient and maintains consistency.

These two problems have motivated the approach to the Flagship system software and to the execution mechanism which supports it.

The architectural style of the Flagship system as viewed by language users and compilers is guided by the Programmers Reference Model (PRM). The PRM aims to provide a common set of high-level concepts which can be used by all languages and application environments, so that interworking between applications and between software components constructed using different languages is facilitated. Therefore, it must specify in abstract form a large part of the user interface to the Flagship system software. The high-level concepts chosen to be represented in the PRM are intended to enable application programmers and designers to take a more abstract view of application design than in third and fourth generation systems, so that the cost of application development is reduced. The Flagship system software has two responsibilities (Broughton et al., 1987):

- The provision of the PRM both on the target machine and on host operating systems, such as UNIX (a trademark of AT&T Bell Laboratories) and VME, the ICL Mainframe Operating System (Warboys, 1980). This part of the system software consists of a "library" of system functions on the machine in addition to functions provided by the host. The host will also provide system management.
- The management of system resources and provision of a secure multi-user environment. A great majority of these duties require support from the hardware or underlying system and will be defined in the Implementation Specific Interface (ISI) particular to the Flagship machine or UNIX/VME host system.

Since the machine is designed to execute declarative languages efficiently and to exploit the parallelism available, the system software is also implemented using a declarative language. Moreover, system software has traditionally

been seen as a notoriously difficult area of software engineering prone to both serious and numerous errors. Given the claimed advantages of declarative languages their use should ameliorate, to some extent, these problems.

The functional language HOPE+ (Perry, 1987a) based on HOPE (Burstall et al., 1981) was chosen as the language of implementation. There are various approaches to implementing operating systems using functional languages but in particular there are two features of operating systems that do not fit naturally onto functional languages:

- **State:** operating systems control access to non-shareable resources by guarding them with mutable variables which have a single instance in the system; functional languages do not have the concept of updatable state.
- **Non-determinacy:** operating systems need to respond to non-deterministic actions, e.g. the typing of a character at a keyboard; programs written in functional languages do not naturally deal with such non-determinism.

There are two major approaches to modelling state in a functional language: continuations (Perry, 1987b) and streams (Henderson, 1982). Non-deterministic behaviour, the ability to respond to events in the order in which they occur, may be supported in a purely functional way by adding information to represent the passage of time (e.g. the **hiatons** approach of Wadge and Ashcroft, 1985). Other approaches are "almost" functional: adding a non-deterministic stream operator, **merge** (Henderson, 1982), which receives two input streams and nondeterministically interleaves them into one output stream (for a survey of these issues see Jones and Sinclair, 1989).

The Flagship approach is that it is considered both more natural and more efficient to express the behaviour of operating systems using a collection of state variables that are updatable over the course of time, and whose value at any instant represents the state of the computation at that instant (Banach et al., 1988). The unit of abstraction, design and decomposition for the system software is the Flagship Abstract Data Type (FADT) (Leunig, 1987, 1988; Boddy, 1988). In the general case, an instance of an FADT holds a state variable accessed via a mechanism which has properties similar to the non-deterministic merge operator.

The present paper examines the declarative and non-declarative aspects of the system software, particularly from the point of view of protection and binding. The paper considers the implementation of FADTs at three levels of abstraction: the execution mechanism level; the PRM (user interface) level; the system software level. In particular the lowest, *kernel*, level of the system software is considered. Mechanisms for binding and for protection are considered at each level. A binding mechanism (the Static Copy Environment Table (SCET)) is introduced which enables efficient local access to code and data, and provides a protection context analogous to a capability list. This mechanism can be used in a purely declarative fashion and may be generalised to handle non-declarative features cleanly and efficiently.

A subset of the kernel has been implemented on the Flagship emulator, which consists of sixteen processing elements (Townsend, 1987), at ICL, West Gorton. However, many of the ideas relating to SCETs and protection discussed in this paper are not supported in that initial prototype.

2 The Flagship basic execution mechanism

In order to discuss the approach taken to the system software it is necessary first to briefly consider the execution mechanism which supports it. The Flagship Basic Execution Mechanism (BEM) (Watson, 1987b) supports both declarative and non-declarative aspects of computation. The features of present interest are those which deal with the problems associated with distribution of a computation over multiple processing elements. These features are:

- a rewrite is atomic: it is guaranteed that a rewrite is not interrupted by another rewrite.
- a rewrite is localised to a single processing element in order to support atomicity: selection of this processing element may be different in the declarative and in the non-declarative case, as will be shown below.

These properties of locality and atomicity are used to provide a mechanism for supporting state, as will be described in section 2.2.

2.1 Declarative aspects

Flagship is a graph reduction machine which executes a subset of DACTL (Glauert et al., 1987) rewrite rules, so that all programs are compiled into DACTL rewrites. The DACTL rewrites are implemented as graphs of packets which are reduced by the execution of the rules associated with the particular rewrite.

A particular programming paradigm (e.g. functional) is modelled in terms of DACTL at the Computational Level (CL). The model for functional programming at the CL is the supercombinator computational model (Watson, 1987c). Beneath this level, the BEM provides the primitives for the execution of actions defined at the CL.

In order to support the CL, the BEM "recognises" various packet types – encoded in their header – with which the BEM associates different actions. Banach and Watson (1988) pointed out that packet types fall into three classes: function, constructor and stateholder. The first two of these classes support a declarative style.

- Function class packets are *application* packets which appear at the roots of rewritable subgraphs. These can contain a pointer to a built-in function or a pointer to a CODE packet which contains executable machine code for some defined function. "Firing" an application packet causes the subgraph

of which it is the root to be rewritten according to some rewrite rule associated with the function.
- Constructor packets hold unalterable base values. They are used to construct, for example, lists, or to hold the result of an evaluation. Constructor packets can be copied, as when a particular constructor packet in the store of one processing element is being accessed by another processing element.

Declarative rewrites, not involving a stateholder, are localised to the processing element of the root packet of the reducible subgraph.

2.2 Non-declarative aspects

In order to deal with state, the BEM supports MONSTR (Banach, 1987), a subset of DACTL, which constrains the execution of rewrites involving stateholder packets. The MONSTR model is designed to permit state to be used in a clean and controlled way while eliminating opportunities for performing side-effects through other mechanisms in the model (Banach and Watson, 1988).

Stateholder packets represent **state variables** which have the properties:

- They cannot be copied, i.e. once a stateholder packet is created it is unique and is located on a single processor-store unit.
- At most one state holder can be involved in a rewrite.

These two properties of stateholder packets, allied with the locality and atomicity properties of a rewrite on the Flagship machine, allow serialized update and interrogation of stateholder packets.

Non-declarative rewrites are localised to the processing element where the stateholder resides.

The execution level constrains, in the general case, the implementation of FADTs in two ways:

- The state of an FADT must be representable as a single stateholder packet.
- All operations on an FADT must be the size and complexity of a single rewrite.

2.3 Flagship environments

The reduction of a graph of packets is performed in the environment of the root packet. The concept of the static environment supports "system aspects" of the computation (Marsh and Leunig, 1988).

- Scheduling controls – process information – the priority of the computation, the number of processing elements required, the limits on store size and processing time for computation.

- Protection domains – the environment defines a "domain" of entities that may be accessed by the computation – a protection domain, referenced by an identifier.
- Name-binding context – there is a need for accessing global and/or local resources in the environment distributed over multiple processing elements. The static environment provides the Static Copy Environment Table – a means of accessing processing element-local copies of code and static data. This SCET may be used either declaratively or non-declaratively and is discussed extensively below.

The environment of a rewrite is established by the BEM as part of the "reduction cycle" during which a subgraph is reduced. It is represented by an environment identifier located in the header of the root packet. This identifier is an index into the environment table of the processing element.

2.4 Binding at the BEM level

An important consideration in distributed systems is the global-to-local mappings that occur. Ideally, given the latency of the network, access to all system facilities should be performed locally. There is thus a notion on Flagship of static copying to provide local copies of static data and code. The SCET mechanism was introduced (by Watson, 1987a) as a way of simplifying the problem of obtaining and referring to (*binding* to) local copies of packets. In addition to providing an efficient copying mechanism for static objects, the SCET concept provides a flexible binding mechanism via an indirection. Having one logical SCET per environment provides a local binding environment where the same SCET index will bind different objects in different environments.

The logical SCET is a flat table used by the execution mechanism to translate SCET pointers into local packet addresses. The implementation of this on Flagship is that for each logical SCET there is a physical SCET on each processing element in the neighbourhood associated with the environment. When the translation of a SCET pointer to a local address takes place in a rewrite, the physical SCET on the processing element performing the rewrite is used.

The SCET mechanism is intended to be a demand copying mechanism, in that copies are made to local SCETs when a rewrite on that processing element accesses a SCET entry. This has both declarative and non-declarative aspects (Keane, 1989).

2.4.1 SCET as a declarative mechanism: As explained above, a mechanism is required which allows copying on demand of code and static data to local stores and sharing of such copies within an environment. A SCET is associated with each environment. Any static information is referred to via an offset into this table (Fig. 1). The addresses held in the SCET are either local addresses within a particular process store, or special NULL values

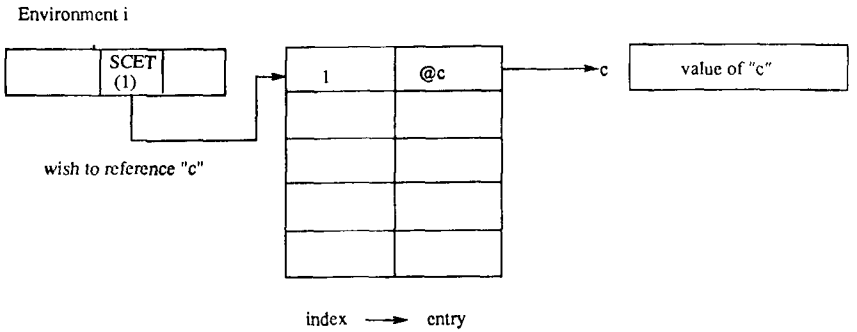


Fig. 1 A logical SCET showing the indirection mechanism. A reference to "*c*" in the environment *i* is resolved via an indirection to an offset in the SCET for environment *i*. The SCET entry at offset 1 contains a pointer @*c* which references *c* so that its value may be accessed.

indicating that no local copies exist. If such a NULL value is encountered then a copy is made from the Master SCET which is located on the "root" processing element of the neighbourhood of processing elements in which the current computation may be executed (Fig. 2). A reference to the appropriate SCET is stored in the environment table entry for this environment. The basic execution mechanism of Flagship does not distinguish between "real" addresses and the "indirect" address which is a SCET index.

These two means of addressing provide alternative ways of linking packets together in a graph. The normal pointer mechanism allows one packet to reference another packet by its global store address (pe, page, packet). The SCET mechanism allows one packet to reference another packet by an index: this index is a pointer into the SCET associated with the current environment, and provides the address of a local copy of the referenced packet.

The architecture allows many SCETs in the machine. The particular SCET used during a reduction is obtained from the Static Environment for the reduction; that is, from the Environment Table entry for the root packet of the reduction. In this way, the root packet of a reducible subgraph can identify the particular SCET with which its execution is to be associated. Since the SCET mechanism is a means of addressing packets, the SCET mechanism provides a binding mechanism which thereby supports the context for a reduction. In the declarative world, such a binding mechanism should be referentially transparent; the same SCET index should always be bound to the same entity.

2.4.2 SCET as a non-declarative mechanism: Having a distributed implementation of a logical SCET allows "nodal variants" to binding where within a single environment, the same SCET index binds to different entities in different processing elements. To further examine the use of the SCET mechanism to access objects, it is possible to consider an FADT instance

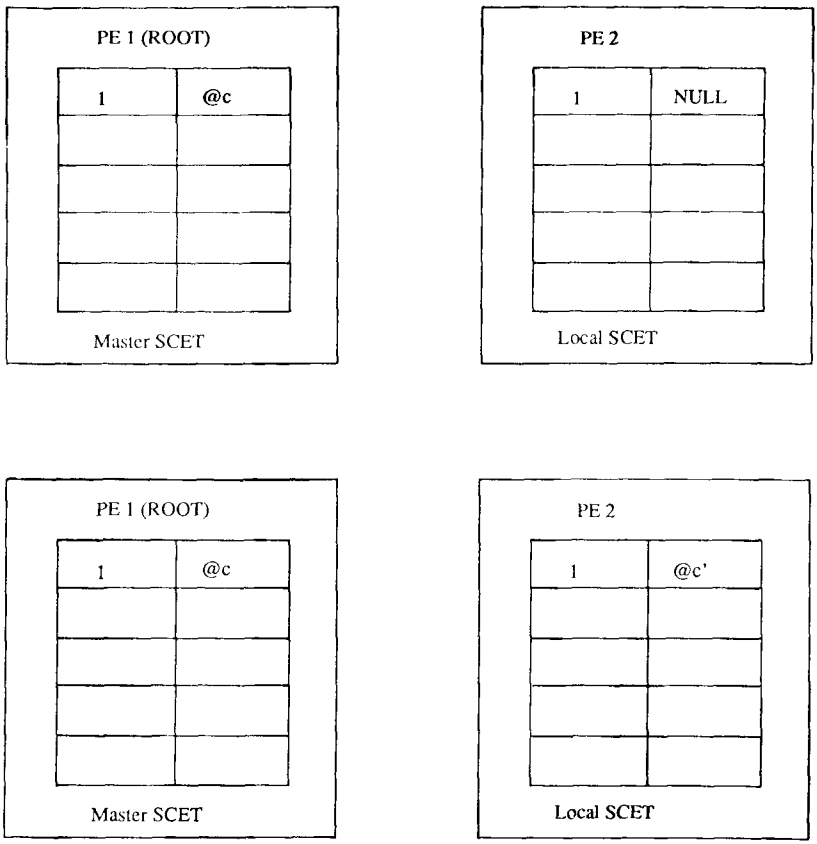


Fig. 2 Physical SCETs showing the effect of the copying mechanism before and after a rewrite involving a reference to an entity *c* via a SCET index *1*. The rewrite is executed on processing element *PE2* and invokes the copying mechanism to obtain a copy of *c* from the root processing element *PE1*.
(a) The Local SCET on *PE2* has a *NULL* entry at index *1*, indicating that no local copy of *c* exists in that processing element. The entry at index *1* of the Master SCET on *PE1* contains the physical address of *c*.
(b) After the rewrite referencing SCET index *1* is fired on *PE2*, a local copy of *c*, *c'*, has been made in the packet store of *PE2*.

representing a **global resource manager**. Such an FADT instance is invoked when there is a need to co-ordinate kernel activity (that is, activity in the lowest level of the system software) across two or more processing elements. The stateholder for such a global manager instance is located on a particular processing element, and consequently must be accessed on *that* processing element. This represents a bottleneck if many requests to this global resource manager are made, as the requests cannot be handled in parallel.

Consequently the resource managers have been structured so that most activity can occur on a local (per processing element) level rather than invoking global (involving at least two processing elements) activity. This has manifested itself in the presence of **local resource managers** on each processing element, each represented by an FADT instance. Given this structuring, when such local resource managers are accessed, the particular instance accessed is the instance on the current processing element. Each of these local instances are addressed via the same index in the local SCET but each local SCET contains a different reference bound to the local resource manager.

This introduces a **nodal variant** SCET. When the SCET entry is assigned, a mapping between processing element and some local instance is used to place the appropriate instance reference in the appropriate local SCET. This is so that the usual SCET copying from the master SCET will be inhibited because of the presence of a local copy before any reference is made to that entry. This nodal variant SCET allows access to kernel interfaces to be done in an efficient manner by avoiding communication across the network.

This method is very similar to the **nodal store** of the distributed VME model (Warboys, 1985) in which identically named locations in every virtual machine have different contents on different nodes. As with the notion of keeping Flagship resource manger instances on a nodal basis, the contents of VME nodal store are not of system-wide interest. In both cases, the nodal variants are not visible outside the "owning" processing element. An executable model for SCETs has been developed by Keane (1989).

2.5 Security/protection at the BEM level

As noted in section 2.3, the environment of each computation defines a "domain" of entities that may be accessed by the computation – a protection domain. Thus, each entry in the environment table contains a protection domain identifier. During the reduction cycle, the current protection domain is established by the BEM from the current environment. This Flagship protection domain is unlike that of "pure" capability machines (Levy, 1984), where the protection domain is implicitly defined by the objects currently accessible to a computation, where these objects are accessed by capabilities. In contrast, the Flagship protection domain is explicitly identified by the protection domain identifier.

Pointer items are tagged. Pointers of type *cap* can form the basis of a protection mechanism. If the root packet of a rewrite references a packet in a different protection domain, then that reference must be via a *cap* pointer. Any reference which breaks this rule must cause an exception. In conventional capability machines, capabilities are protected either by tagging or by restricting them to capability segments (Levy, 1984). The presence of a *cap* tag in a pointer item of a packet transforms that packet item into a *capability* for the referenced packet. However, conventional capabilities are a pair $\langle AR, ID \rangle$ (Corsini et al., 1984) rather than the $\langle ID \rangle$ of the *cap* pointer item. This point is dealt with further at the kernel level (section 4.4).

3 Flagship PRM

The PRM (Thomson, 1987) is based on declarative programming, using graph reduction as the computational mechanism. It is however recognised that most real applications are not declarative; they need to represent entities which have changeable state. The PRM deals with this directly by adding ideas of state and change to the base idea of declarative programming.

The smallest entity which can have state is called an **object**; the action of changing the state of an object is called an update. Introducing these ideas forces a change on the computational mechanism, i.e. it is necessary for the user to have control over whether a particular reduction is performed once or many times and over the order in which reductions are performed. Additionally, the reduction mechanism should be capable of producing different results on different reductions of the same subgraph.

Clearly it is necessary to introduce a concept that deals with the concept of objects and update in a controlled fashion. This concept is the **atomic action** which packages up a collection of non-declarative actions into a single operation which is effectively atomic, i.e. either all of it has happened or none of it has. This atomicity implies that a transaction cannot be affected by partial completion of another atomic action (Broughton et al., 1987).

FADTs above the kernel of the system software level are implemented using this notion of objects and atomic actions. Objects and atomic actions allow multiple operations on a single FADT and or on multiple FADTs to be serialised. Objects (of some form) and atomic actions are available above the kernel.

An executable model (Sa, 1987) and an operational semantics (Hussak, 1988) have been developed for a substantial section of the PRM.

3.1 Security and binding at the PRM

As described earlier the PRM provides a model of objects and atomic actions upon them. The security system is based upon this model. Objects are accessed by computations. Computations occur within a process and inherit the access rights of their owning process. A process receives its access rights from the "principal" object under whose auspices it is run. A principal object is the system representation of a user.

The access to objects is regulated in two ways (Thomson, 1987):

- **statically:** the static security system controls the kind of access which is permitted to each object. For any two objects a and b, there is a set of operations s that computations which derive their access rights from a are allowed to perform on object b. When an operation o on object b is requested by a computation, a check is made to see whether $o \in s$.

- **dynamically:** the dynamic security system has been based upon the US Department of Defense "B2" class (DoD, 1983). A set of classifications is used to limit the flow of information from highly classified objects to objects of lower classification. Each object in the system has a classification, and each process has a current classification and an upper limit classification.
  All computations are restricted to obtaining information only from objects with a lower classification than the current classification of the process at which it is rooted. If the accessed object's classification is higher than the processes current classification then the processes current classification is set to the object's classification.
  Computations can only write to or destroy objects that have a classification lower than the current classification of the process at which it is rooted. It is possible to override the classification check in these circumstances if the static access permissions include a **write down** operation.

Security can be considered as dynamic binding. Both aspects of the security system entail the possibility of the allowable operations on an object by a process changing, either because of changes in the access rights or by a change in the classification of either the object or the process. This can be seen as a dynamic binding between objects and the operations applicable to them within a particular process. A VDM (Jones, 1986) specification of most aspects of the PRM security system has been given by Keane (1987).

There is a further aspect of binding at the PRM level: PRM objects may be said to be bound together (Mayes, 1988b) in that they may be accessed by, and will thus participate in, the same atomic action.

4 Flagship kernel

Having briefly reviewed the execution mechanism and the user-level PRM, the system software can now be discussed. The system software on Flagship has been viewed as a hierarchy of layers, where each layer is composed of instances of Abstract Data Types (Leunig, 1988). FADTs allow concurrent access to their interface operations, and are state-based.

An FADT is a template for a software entity that forms the basic building block from which the system software of the Flagship machine is built. It is important to note the use of the word "template" in that an FADT is simply a description of how FADT instances will appear and behave. This is analogous to how a type definition in a programming language is not itself an instance of its type. An FADT consists of a public part and a private part. The public part is the user's view of the FADT and comprises the user's interface to the FADT; a set of operations which the FADT instance will perform and which are called interface functions (or more strictly, pseudo-functions – the interface function is not a pure function because its output is not wholly dependent on its input). The private part comprises the body of

the FADT instance which consists of any state it controls together with the code to implement the interface functions.

To the extent described above FADTs appear similar to classes in object oriented languages such as Smalltalk (Goldberg and Robson, 1983). The methods of a Smalltalk object are analogous to the interface functions of a Flagship FADT instance. The difference between the two structures is a result of the fact that FADTs must work in a parallel processing environment and are therefore given the extra responsibility of having to deal with the synchronisation issues thus introduced. For instance, it is conceivable that an interface function of a specific FADT instance may be used before another use of the same interface function of the same FADT instance has completed. It is the FADT's responsibility to deal with such concurrency in a well defined and computationally useful manner.

FADTs are non-declarative entities and have the following properties (Leunig, 1987):

- encapsulated state. Each instance of an FADT can have its own state. The state is encapsulated in that access to it, from outside the FADT, can only occur via the interfaces provided by the FADT. State is introduced as part of the basic building block. Its encapsulation is based on the principle of information hiding.
- A "state transition function" style of definition. The general form of an FADT operation will be f: $S \times I \rightarrow S \times O$, where f is the operation, S is the type of the state of the FADT, I is the type of the input parameters and O the type of the results of the operation.
- Procedure call interfaces. The interfaces provided by FADTs will be procedural in style. The procedural style is considered to be a cleaner and simpler style compared to the other alternatives such as message passing; in particular it allows transparency of location within the Flagship machine. The systems that may be used as hosts, e.g. VME and UNIX, are both procedure call based. For the state transition function above, an interface to the function would be f': $I \rightarrow O$.
- A mechanism to ensure the consistency of state.

The implementation of these features is:

- State is referrenced via a pointer to a stateholder packet.
- Procedure call interface is via a rewrite.
- Consistency of state is ensured by a "guardian mechanism".

The mechanism supporting the FADT is built on the MONSTR model. More specifically, FADTs at the kernel layer of the system software are implemented using **guardians**, where a guardian is supported by the MONSTR execution mechanism. Guardians are a technique which build a stream processing function from a state transition function, and then hides the stream from the interface seen by users (Broughton et al., 1987). Guardians are described in the next section.

4.1 The guardian mechanism

Guardians are the lowest level entities in the Flagship software environment for implementing serialisability.

The term guardian in Flagship is used with reference to an implementation mechanism which is used to realise certain types of FADT. It is essentially the concept of guardian introduced by Dennis (1981), and has many similarities to the concept of managers given by Arvind and Brock (1984). It differs from the usage of Liskov and Scheifler (1983) where the term is used to describe the encapsulation of a physical processing element and its associated resources.

To its users a guardian appears identical to any other FADT; it presents to them an interface consisting of a set of pseudo-functions and has all the other properties of an opaque abstract type as described for FADTs. Indeed a distinction between guardians and other FADTs need only be drawn when it is necessary to consider their implementations. Each pseudo-function is termed a *currency*, as described in the next section.

A Flagship guardian controls access to its encapsulated state by treating it as a single non-shareable resource and by serialising access to it. This is done by effectively allowing only one currency application of the guardian to be active at any one time and waits until the currency has completed its manipulation of the state before allowing any other currency application to proceed.

The guardian mechanism is the piece of code which does the work of the guardian. There are several types of guardian mechanism which determine the behaviour of the guardian which they implement. Although all guardians present the same currency style interface to their users, the user can expect different scheduling policies for currency applications depending on the guardian mechanism employed, e.g. a queue guardian ensures fairness of handling requests to the guardian but does not allow pre-emption of a low priority process by a high priority one. A pool guardian on the other hand does allow pre-emption but cannot guarantee fairness. In terms of implementation a guardian mechanism is simply the code body of a CODE packet whose execution makes available, to the state transition function, the state component of that currency application – though this availability of the state value is controlled by a queueing or pooling mechanism. Details of the implementations of both a queue and pool guardian can be found in Holdsworth (1988).

In essence, the guardian mechanism relies on the BEM "locality rule" to enforce sequential access to the state. Since all accesses to state must occur on the processing element of the stateholder, all guardian mechanism executions on the same stateholder must occur on the same processing element and must thus be serial.

Access to the guardian is essentially by a non-deterministic merge of the requests to its currencies (Fig. 3). The state of any FADT can be viewed as being a special case of a stream in that it can be viewed as an infinite list evaluated as and when the currencies are rewritten.

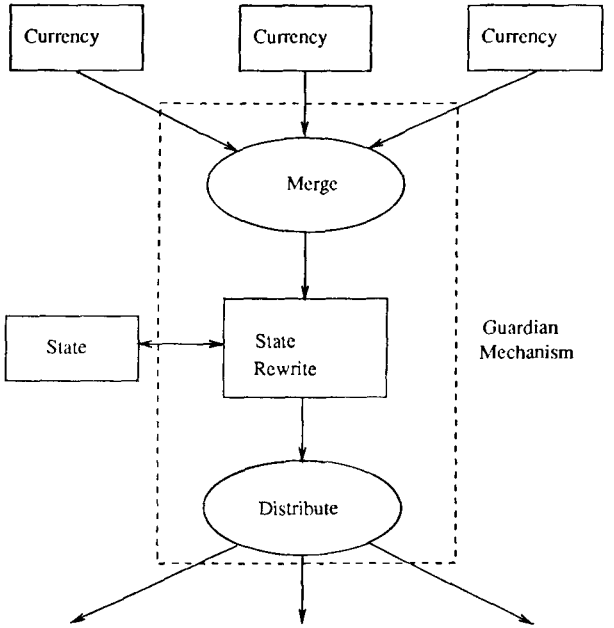


Fig. 3 A guardian as a non-deterministic merge of currency applications.

4.2 Implementation of kernel FADTs

The state of an FADT instance is represented by a stateholder, *SH*. The representation of the state of an FADT instance may be accessed only by operations (state transition functions, *stfs*) defined for the FADT. However, not only is the state representation inaccessible to the caller, but so are the *stfs*. The only visible "handles" on an FADT instance are a set of interface functions (*iffs*) which have no explicit reference to the state in their signatures. It is these *iffs* which are accessible to users of the FADT instance. The transformation of an *stf* into an *iff* is achieved internally by a **guardian mechanism** (*g*). This accesses the current state value in the stateholder, allows the application of stf to this value, updates the state in the stateholder with the new value and delivers the result of the stf to the user. This mechanism ensures that access to the FADT instance is serial. Thus an FADT instance in Flagship kernel may be said to consist of a set of iffs all of which refer (internally) to the same stateholder (Fig. 4).

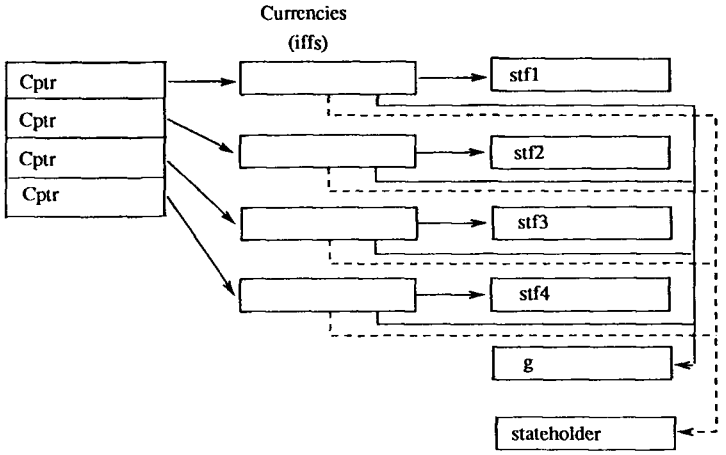


Fig. 4 The implementation of an FADT instance. Interface operations (iffs) are implemented as currencies. Each currency refers to the same stateholder (SH) and links to it a guardian mechanism (g) and a specific state transition function (stf). Currencies are referenced B Cptrs.

Each iff is a physical linking (*g*, *stf*, *SH*, *.pp*), where *.pp* represents invariant parameters to the *stf*. Thus, for each FADT instance there is a set of *iffs*, each referring to the same *SH* (the state of the FADT instance) and each *stf* representing an allowed operation on that FADT. A pointer or a reference to the *iff* is termed a *Cptr*. To make a "call" on an *iff*, a linking (*Cptr*, *.in*) is required, where *.in* represents the parameters of the call (Mayes, 1988a). The chosen implementation of an *iff*, that is the linking (*g*, *stf*, *SH*, *.pp*), is a **currency**, the term being borrowed from VME. The rewrite representing the call (shown diagrammatically in Fig. 5) is:

$$\text{call}_{(Cptr,.in)} + \text{currency}_{(g,stf,SH,.pp)} \rightarrow$$
$$\text{currency–application}_{(g,stf,SH,.pp,.in)} + \text{currency}_{(g,stf,SH,.pp)}$$

In terms of packet types, the currency is implemented as an application packet without its full arity of arguments; that is, the *stf* requires the input parameters of the call, *.in*.

The **call** to the currency is implemented as an application of an unknown function; that is, the functions within the currency are hidden from the call packet. In functional language terms, this represents the implementation of a higher order function. Thus, FADT instances communicate by calling each other's operations, and this communication is implemented as a rewrite involving these packets. The currency packet may persist, awaiting the next call.

The currency application (*g*, *stf*, *SH*, *.pp*, *.in*) is an invocation of an operation of a particular FADT instance.

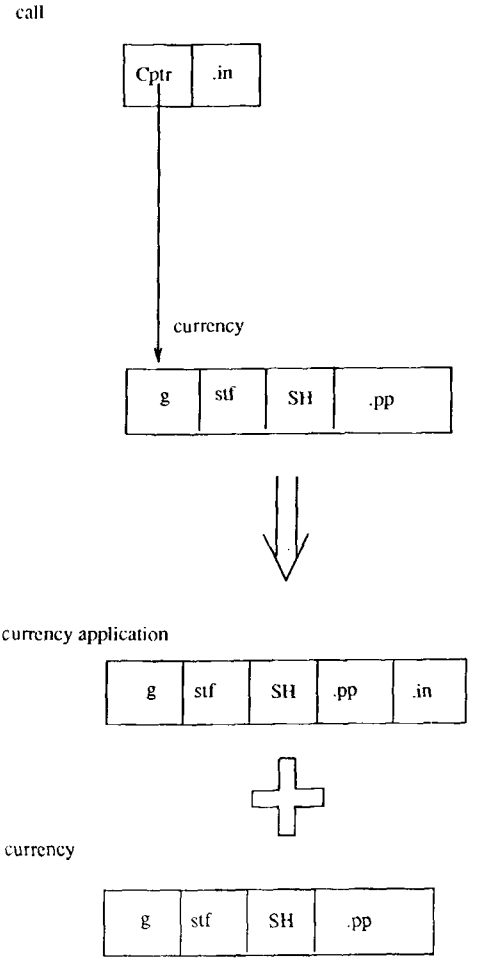


Fig. 5 Packets involved in a call to a currency. Note that the currency application contains both the invariant partial parameters .PP and the parameters specific to the call, *.in*. See legend of Fig. 4 for the meaning of the other symbols.

4.3 Binding in the kernel

4.3.1 Linking in currencies: As shown in the previous section, a currency consists of code and data linked together. There are two reasons for this linking:

- The computational model of Flagship provides a mechanism for the packet implementation of a currency. A currency may thus be implemented as a packet which consists of items referring to the guardian code, the state transition function and the stateholder, and any parameters

whose values are the same for each call to the currency. The remaining parameters, whose values are unique for each call to the currency, are provided by a call packet.

- The concept of currency in Flagship has been (apparently) influenced by analogy with the VME currency. This latter provides not only a way of naming an object but also describes the object's "correct" behaviour (Warboys, 1980). This implies that permitted operations are *implicitly* partially parameterised by the object. Accordingly a Flagship currency is an operation with an implicit object associated with it. Upon instantiation of the Guardian which implements a Flagship FADT instance, an stf is partially parameterised with a pointer to the state as the bound argument. This partial linking enhances both efficiency and security. Efficiency is increased with the advantage of eager linking – linking need not occur at call-time. Security is increased because the user has a reference to a *specific* operation; this is further discussed in section 4.4.

4.3.2 Binding between currencies to support FADT instances: Binding can be described as the act of choosing a specific lower-level implementation of a particular higher-level semantic construct (Saltzer, 1978). Thus, the components of a particular kernel FADT instance are bound together as a set of iffs, where each iff is implemented as a physical linking called a currency. Currencies implementing an FADT instance are bound together in that they share the same stateholder. Even in the case where there might be multiple copies of a currency implementing a particular FADT instance distributed over the system, the BEM "locality rule" ensures that any rewrite involving such a currency copy must be exported to the processing element on which the referenced stateholder, and by implication the FADT instance, resides.

4.3.3 Binding between FADT instances: FADT instance *FADT1* communicates with FADT instance *FADT2* by executing a call to an *FADT2* currency. Thus the "access domain" (which on Flagship is distinct from the explicitly-defined protection domain) of instance *FADT1* – the set of FADT instances with which *FADT1* may communicate – is represented by that set of *Cptrs* which *FADT1* may access.

This is similar in structure to the capability-list of capability machines (e.g. HYDRA, Wulf et al., 1981). The capability-list provides a naming context such that terse local names, indices into the capability structure, can be used to name objects (Jones, 1978).

The set of Cptrs available to an FADT instance may be considered to be a context component of the environment of an operation invocation. Thus the basis of gaining context information is being in the correct environment. Virtual addresses or local names may be resolved with respect to the context component of the current environment. By switching environments, context for binding virtual addresses/local names to Cptrs will change.

If a Cptr-list could provide the context of an invocation of an FADT instance operation, and context is part of the environment of the FADT instance, then the Cptr-list could reside in the static copy environment table (SCET) of that environment. This would mean address/name binding to Cptrs by SCET entries.

This is a further non-declarative use of the SCET enabling FADT instance-specific name-binding. This use raises the issue of whether the currencies *to be accessed by* resource managers can be placed in the SCET. Although there is obvious inefficiency involved in making copies of currencies to processing elements where they cannot be used, there is no necessity to restrict this copying since the guardian/MONSTR mechanism ensures that the currency cannot be activated on any but the processing element where the stateholder referenced by the currency resides.

4.3.4 Kernal objects: As has been discussed in section 3, **objects**, referenced by an object identifier, are the smallest entity at the level of the PRM that can have state associated with them. This state can be changed during the lifetime of the object. At the PRM level, object manipulation is restricted to occur only in the context of an **atomic action**. In order to implement the PRM-level atomic action there must be a set of primitives available which can be used to manipulate objects directly. These primitives will still need to access objects in a serialisable manner if constructive concurrency is to be supported. To this end a set of primitives needs to be supplied at the kernel level which support the PRM level.

FADT instances are bindings between currencies which consist of eagerly linked code and data. However, the FADT mechanism is rather heavyweight for use in supporting lightweight objects which may be short-lived. A more realistic solution is to regard each object identifier as a pointer to a stateholder packet which contains the type and value of the object. The view of the system being composed out of a hierarchy of FADTs can then be maintained by providing a new FADT which will not have any associated controlled state. This FADT will provide as currencies all the necessary manipulation functions on objects including the ability to create new objects. Since this FADT does not have any associated state, it can exist in many instances on the system. The only difference between this implementation of objects and that of any other guardian based-FADT is the time at which the stateholder is linked into the currency.

4.4 Security/protection in the kernel

On Flagship, the issue of protection domains can be regarded as being orthogonal to the issue of context switching. This is because an environment in Flagship carries resource, protection and context information as separate components of the environment. This is in contrast to conventional capability systems where the current protection domain and context of a process are both defined by the current capability-list of that process.

Currencies on Flagship may provide a basis for name resolution/address binding but do not support protection domains. The traversal of the boundary of a protection domain is controlled by "capability packets" which depend on the secure implementation of pointer and store management; however, this protection model is not implemented on the emulator (Banach et al., 1988).

As noted in section 2.5, pointer items are tagged, and pointers of type *cap* can form the basis of a protection mechanism. The *cap* pointer tag transforms a pointer item into a capability ⟨ID⟩. However, unlike the conventional capability ⟨AR, ID⟩ (Corsini et al., 1984) there is no access rights component. However, a *cap* pointer item referencing a currency represents a capability for a particular iff: only the stf referenced by the currency may be applied to the state of the referenced FADT instance. Thus, the access right (for the allowed operation) is *implicitly* defined. The early/eager linking in currencies serves to increase security. A Flagship currency, like a VME currency, represents an access to a single operation on a particular instance of some entity.

A call on a currency represents a call on an interface between two FADT instances. The rewrite representing that call consists of a call packet (at the root of the rewrite) whose environment is that of the caller-FADT instance, and a currency packet (and its child packets) whose environment is that of the called-FADT instance. A call on a currency thus represents an environment switch; this switch may also represent a change from one protection domain to another. In such a case, the Cptr item of the call packet must be a *cap* pointer (Mayes, 1989).

5 Summary

Flagship is a distributed declarative machine. This imposes two problems for the system software: sharing resources in a distributed world and handling state in a declarative world. The problems associated with multiple processing elements and state are solved by the Flagship execution mechanism and the MONSTR language it executes. The Flagship guardian, based on these low-level mechanisms, provides a means for implementing serialisability in the software environment. Kernel FADTs, which represent the unit of abstraction and design for the system software, are based on guardians. The execution of FADT instance code occurs within a static environment. This environment provides protection information and name-binding information. Protection is based on explicitly identified protection domains and, although not implemented, would be based on capability pointers. This mechanism, plus the currency mechanism used to implement the interface functions of FADT instances, is discussed with reference to the "capabilities" of capability machines. Name-binding is provided by the SCET; this can support both declarative or non-declarative contexts: that is, binding contexts which may or may not be referentially transparent over a physically distributed environment.

6 Acknowledgements

The authors would like to thank Professor Brian Warboys and the Flagship teams at Manchester University and ICL, West Gorton. Steve Leunig, the system software designer, Tom Thomson, the PRM designer, and Steve Prior originated many of the ideas presented in this paper, though they cannot be held responsible for any of our interpretations of them. Flagship is an Alvey project, IKBS 049. This work was supported by SERC grant GR/E 21070.

References

ARVIND and BROCK J.D. (1984). 'Resource Managers in Functional Programming'. Journal of Parallel and Distributed Computing 1, 5–21.

BANACH, R. (1987). 'Formal specification of MONSTR'. Flagship document FS/MU/RB/010-87, University of Manchester.

BANACH, R. and WATSON, P. (1988). 'Dealing with State on Flagship: The MONSTR Computational Model'. CONPAR 88, UMIST, Manchester.

BANACH, R., SARGEANT, J., WATSON, I., WATSON, P. and WOODS, V. (1988). 'The Flagship Project'. IEE/BCS UK IT 88, Conference Publication, University College, Swansea, 4–7 July 1988, 242–245.

BODDY, G.S. (1988). 'The Use of VDM within the Alvey Flagship Project'. VDM – The Way Ahead, R. Bloomfield, L. Marshall, R. Jones (eds.), Lecture Notes in Computer Science 328, 153–160.

BROUGHTON, P., THOMSON, C.M., LEUNIG, S.R. and PRIOR, S. (1987). 'Designing System Software for Parallel Declarative Systems', The ICL Fifth Generation Programme, Special Issue of ICL Technical Journal 5, 541–554.

BURSTALL, R.M., MacQUEEN, D.B. and SANELLA, D.T. (1981). HOPE: 'An Experimental Applicative Language'. Proceedings of 1980 LISP Conference, Stanford, California, 136–143.

CORSINI, P., FROSINI, G. and LOPRIORE, L. (1984). 'The implementation of abstract objects in capability-based architecture'. Computer Journal 27, 127–134.

DENNIS, J.B. (1981). 'Data Should Not Change: A Model for a Computer System'. Laboratory for Computer Science, CGS Memo 209, MIT Cambridge, Mass.

DoD (1983). Department of Defense trusted computer system evaluation criteria. CSC-STD-001-83, Department of Defense Computer Security Centre, Maryland.

GLAUERT, J.R.W., KENNAWAY, J.R. and SLEEP, M.R. (1987). 'DACTL1: A Computational Model and Complier Target Language based on Graph Reduction'. The ICL Fifth Generation Programme, Special Issue of ICL Technical Journal 5, 509–537.

GOLDBERG, A. and ROBSON, D. (1983). 'Smalltalk-80, The Language and its Implementation'. Addison-Wesley.

HENDERSON, P. (1982). 'A Purely Functional Operating Systems'. Functional Programming and its Applications, J. Darlington, P. Henderson and D.A. Turner (eds.), Cambridge University Press, 177–189.

HOLDSWORTH, S. (1988). 'The Implementation of Layers of Abstraction of Serialisability in a Parallel Packet Rewrite Machine'. Transfer Report, Department of Computer Science, University of Manchester.

HUSSAK, W. (1988). 'Specification of the PRM'. Flagship Document, Department of Computer Science, University of Manchester.

JONES, A.K. (1978). 'Protection mechanisms and the enforcement of security policies'. Lecture Notes in Comp. Sci. 60, 228–250.

JONES, C.B. (1986). 'Systematic Software Development Using VDM'. Prentice-Hall International.

JONES, S.B. and SINCLAIR, A.F. (1989). 'Functional Programming and Operating Systems'. The Computer Journal 32, 162–174.

KEANE, J.A. (1987). 'A VDM Specification of the PRM Security'. Flagship Document, Department of Computer Science, University of Manchester.

KEANE, J.A. (1989). 'Aspects of Binding in a Declarative System'. MSc. Thesis, Department of Computer Science, University of Manchester, in preparation.
LEUNIG, S.R. (1987). 'Abstract Data Types in the Flagship System Software'. Flagship Document FLAG/DD/303.xxx, ICL.
LEUNIG, S.R. (1988). 'Design Description of the System Software'. Flagship Document FLAG/DD/3SD.016, ICL, with draft changes August.
LEVY, H.M. (1984). 'Capability-based computer systems'. Digital Press.
LISKOV, B. and SCHEIFLER, R. (1983). 'Guardians and Actions: Linguistic Support for Robust, Distributed Programs'. ACM Transactions on Programming Languages and Systems 5, 381–404.
MARSH, A.H. and LEUNIG, S.R. (1988). 'Flagship Emulator ISI Definition'. Flagship Document FLAG/DD/3SD.018, ICL.
MAYES, K.R. (1988a). 'Currency as a Level of Abstraction in Flagship Kernel'. Flagship Document FS/MU/KM/022-88, University of Manchester.
MAYES, K.R. (1988b). 'A currency view of Flagship ADTs and objects'. Flagship Document FS/MU/KM/023-88, University of Manchester.
MAYES, K.R. (1989). 'Capabilities and Protection domains on Flagship'. Flagship Document, University of Manchester.
PEYTON JONES, S.L. (1987). 'The Implementation of Functional Programming Languages'. Prentice-Hall International.
PERRY, N. (1987a). 'HOPE+'. Internal Document IC/FPR/LANG/2.5.1/7, Department of Computing, Imperial College, London.
PERRY, N. (1987b). 'HOPE+C'. Internal Document IC/FPR/LANG/2.5.1/21, Department of Computing, Imperial College, London.
SA, J. (1987). 'Interpreting PRM Expressions in a Functional Environment'. Flagship Document, Department of Computer Science, University of Manchester.
SALTZER, J.H. (1978). 'Naming and binding of objects'. Lecture Notes in Comp. Sci. 60, 99–208.
THOMSON, C.M. (1987). 'PRM Definition'. Flagship Document FLAG/DD/101.001, ICL.
TOWNSEND, P. (1987). 'Flagship Hardware and Implementation'. The ICL Fifth Generation Programme, Special Issue of ICL Technical Journal 5, 575–594.
WADGE, W.W. and ASHCROFT, E.A. (1985). 'Lucid, the Dataflow Programming Language'. APIC Studies in Data Processing No. 22, Academic Press.
WARBOYS, B.C. (1980). 'VME/B A Model for a realisation of a Total System Concept'. ICL Technical Journal 2, 132–146.
WARBOYS, B.C. (1985). 'VME Nodal Architecture: a Model for the Realisation of a Distributed System Concept'. ICL Technical Journal 4, 236–247.
WATSON, I. (1987a). 'Environments and Contexts etc.'. Flagship Document FS/MU/IW/013-87, Department of Computer Science, University of Manchester.
WATSON, P. (1987b). 'The Flagship Basic Execution Mechanism'. Flagship Document FS/MU/PW/018-87, Department of Computer Science, University of Manchester.
WATSON, P. (1987c). 'A Super-Combinator Model of Computation'. Flagship Document FS/MU/PW/021-87, Department of Computer Science, University of Manchester.
WATSON, P. and WATSON, I. (1987). 'Evaluating Functional Programs on the Flagship Machine'. Functional Programming Languages and Computer Architectures, G. Kahn (ed.), Lecture Notes in Computer Science 274, 80–97.
WATSON, I., WATSON, P. and WOODS, V. (1986). 'Parallel Data-Driven Graph Reduction'. pp. 1–18 of Fifth Generation Computer Architectures, J.V. Woods (ed.), North-Holland, IFIP.
WATSON, I., SARGEANT, J., WATSON, P. and WOODS, J.V. (1987). 'Flagship Computational Models and Machine Architecture'. The ICL Fifth Generation Programme, Special Issue of ICL Technical Journal 5, 555–574.
WULF, W.A., LEVIN, R. and HARBISON, S.P. (1981). 'Hydra/C.mmp An experimental computer system'. McGraw-Hill Book Company.

HISTORY OF ICL

ICL Company Research and Development Part 3: The New Range and Other Developments, 1968–85

Martin Campbell-Kelly
Department of Computer Science, University of Warwick, Coventry CV4 7AL

Abstract

This paper describes the principal R&D activities in ICL from its formation in 1968 up to the STC takeover in 1984. For much of this period, R&D was strongly focused on the development and subsequent enhancement of a new range of computers, launched as the ICL 2900 series in 1974, and relaunched as Series 39 in 1985. The development of the new range made enormous demands on every division of ICL, and caused the company to neglect other aspects of the computer business, such as small business systems and networks. Consequently, ICL's emphasis on mainframes left it ill-prepared to meet the challenges of the 1980s. The paper concludes with a discussion of the management changes and the new product and marketing strategies that helped to achieve ICL's recovery during 1981–84.

1 Introduction

As detailed in an earlier paper, the creation of ICL in 1968 was something of a political act[1]. The company was set up as the national-flagship computer firm with the general mission to compete with the American mainframe manufacturers in the international market place, and with the specific mission to develop a new range of computers for the 1970s. In order to understand ICL's development in the 1970s and 1980s, it is necessary to understand the motives for its creation and its special relationship with the government.

In October 1964, Harold Wilson's Labour Government had come into office convinced that "if action were not taken quickly, the British computer industry would cease to exist"[2]. There were at that time two dominant British computer firms: International Computers and Tabulators (ICT) and the computer division of English Electric. The government view was that it was imperative for ICT and English Electric to merge their computer interests, so that they could combine their resources to meet the "American challenge". There was, however, a major impediment to such a merger: this

was that both companies had independently launched a full range of third-generation computers in 1964–65. In the case of ICT, this was the 1900 series launched in September 1964. The 1900 series was considered to have a somewhat old-fashioned architecture, although it later proved highly successful in the marketplace. In the case of English Electric, it had launched its System 4 range in September 1965. System 4 was based on the RCA Spectra 70 range of IBM System/360 compatible computers, and although it had a modern architecture the policy of selling IBM-compatible equipment was always a controversial one. In 1965, both firms were irrevocably wedded to their respective third-generation computer ranges and marketing strategies, so that it seemed that a merger could not be contemplated until the time came to develop a new range of computers in the 1970s.

By 1967, however, the climate of opinion within both companies had begun to change. In particular, English Electric, which had sustained heavy losses developing System 4, was keen to withdraw from direct involvement in the EDP-computer business. And within ICT, there was a body of opinion which considered that a concentration of the British computer industry was vital if it was to survive the 1970s. As an inducement for the companies to merge, the government held out the prospect of a non-repayable grant of the order of £25–30 million towards the cost of developing a new range of computers. This offer had the effect of precipitating the merger, although the amount of money the government eventually provided was only £13½ million[3]. Consequently, for ICL the scene for the 1970s was set by a mission to develop a new range of computers, but with R&D resources that were not really sufficient.

2 The New Range Planning Organization

ICL began operating as a legal entity in September 1968, and within a month or two serious thought was being given to the new range. It was always realized that funding a new range would be difficult, and initially attempts were made to share the cost with another mainframe manufacturer. In November 1968, discussions were held with several United States firms, including RCA, CDC and Burroughs, to see if it would be possible to find a ready-made solution for the new range, or if a joint development would be possible. None of these discussions produced any worthwhile result, however, and so it was decided that ICL would have to develop the new range entirely from its own resources.

The specification of a completely new range of computers was a once-in-a-lifetime opportunity, and inevitably the protagonists of the various architectural solutions within ICL, and without, were anxious to see their particular vision implemented. In particular, one faction within ICL was proposing John Iliffe's Basic Language Machine (BLM) architecture, while other people were arguing for Manchester University's MU5 computer architecture. These were powerful intellectual and emotional positions. In order to defuse the situation, and to avoid emotional commitment from determining the new

range architecture, it was decided to create a formal planning structure to try to arrive at a rational solution.

The New Range Planning Organization (NRPO) was formed in January 1969, and the specification of the new range was to occupy most of that year. During the first phase (January to April) the possible options for a new range architecture were evaluated in the context of an assumed marketing environment. In the second phase, which occupied the remainder of 1969, detailed specifications for the selected architectural option were elaborated and implementation and introduction strategies explored.

The nucleus of the NRPO consisted of ICL planning staff (from both ICT and English Electric), who had expertise in corporate planning, market research and technical product planning. To this nucleus, staff with appropriate expertise were added – manufacturing personnel, hardware and software designers, sales staff, and so on. Altogether, some sixty people, and fifteen staff-years of effort, went into the first phase of new range planning during the first quarter of 1969. In addition, about a dozen senior academics were retained as part-time consultants. The NRPO was divided into a dozen or so small teams, none of them more than six people in size. These study groups were broadly classified as "aims", "options" and "assessment" teams. The function of the "aims" teams was to determine the market environment from the mid-1970s up to the mid-1980s. The job of each of the "options" teams was to investigate in detail one of several possible architectural options for the new range. Finally, the task of the assessment teams was to determine criteria by which to measure the options, and to independently identify and specify key criteria to be met by the new range – for example the need for "bridging" techniques to ease transition from the current ranges, the need for resilience and security, and the need for advanced database and compiler techniques.

The first major assumption of the marketing environment in which the new range would exist, was that IBM would dominate it, just as it had in the 1960s, with well over 50 per cent of the world market. On the other hand, no assumptions could be made about IBM's product strategy: its "future series", expected in the early 1970s, might be an enhancement of the System/360 architecture, or it might be an entirely new architecture with emulation and other aids to assist the migration of users. A second major assumption of the future computer market was that there would be a transition from primarily batch-processing computers to primarily real-time transaction processing systems. In evaluating the success of the new range it is important to understand the uncertainties of the market environment for which it was designed. As it happened, IBM did not change its machine architecture (and at the time of writing still has not). Also, although large real-time computers were a major growth area in the 1970s, a much bigger growth occurred in small decentralized computers for which the 2900 series was not well adapted (nor was System/360).

3 New range: the strategic options

Altogether, seven "option" teams were established to evaluate different architectural options for the new range, the only constraint being the assumed marketing environment. Although there were seven options, they in fact amounted to three basic choices: first, to enhance one of the current ranges; second, to use an existing advanced architecture; and third, to develop a new architecture. In fact the third choice was taken; the arguments for this choice, are detailed below.

1 Options 1 and 2: enhancement of current ranges

The most conservative and cheapest choice was to enhance one of the current ranges, the 1900 series or System 4. The "8-bit 1900", as it was called, was under active study in ICT long before the merger, both by an internal team and externally by the Auerbach consultancy. The three main problems with the 1900 series were its 6-bit character/24-bit word, its lack of real-time capability, and the fact that it was perceived as old fashioned. The first was easily put right, and both study teams recommended a 32-bit word architecture downwards compatible with the 24-bit 1900. Enhancing real-time capability was also possible, although this was less straightforward. But the 1900 series was an ageing architecture and there was no real prospect of sustaining it beyond the mid-1970s. Consequently there was little support for 1900 series enhancement, other than for short term tactical purposes while some longer term range was under development.

System 4 had real merit, even without enhancement, as ICL's new range. It already used an 8-bit IBM-compatible byte, and it had real-time facilities that were much better than those of System/360. Moreover, a good deal of work had been done on future development within English Electric, and ICL had access to RCA's new product line, then under development to replace Spectra 70. The System 4 option consequently had strong support, especially from the English Electric faction within ICL, and the technical arguments were very strong. The principal argument against a 360-based architecture was whether ICL should go IBM-compatible: the case against IBM compatibility was to prevail in ICL, just as it had in ICT.

2 Options 3–6: existing advanced achitectures

If ICL was to adopt a new architecture, then there was some merit in using an existing architecture, which would be less of a leap into the dark than developing its own *ab initio*. In fact, there was no architecture to which ICL had access that met all its requirements, so that the four architectures examined served largely as inputs to the "synthetic option" (see below). The four architectures studied were the "New Series Branch" option, the High-Level Language option, the Manchester University option, and the Basic Language Machine (BLM) option.

The New Series Branch option began, in effect, as a shopping list from ICT software writers of desirable features in a new range. It was a minor input to the synthetic option, but was otherwise something of a paper tiger. The High-Level Language option was to adopt the Burroughs' design philosophy of a machine oriented to a particular high-level programming language. On its own, this design philosophy was insufficient for the market of the 1970s, but as with the MU5 architecture (below) the need to execute high-level languages efficiently was subsumed in the synthetic option.

Manchester University had had very close links with ICL (via ICT and Ferranti) going back many years, so it was natural that ICL would seriously consider its latest machine, the MU5, then coming into service as a successor to the Atlas[4]. Designed for a "number crunching" environment, the architecture was insufficient as the sole basis of a new range, but it contained many useful innovations and was one of the two major inputs to the synthetic option. Five of the Manchester University faculty held consultancies on the design of the new range.

ICL's own BLM project[5] was the second major inspiration for the new range, for it resolved in a very generalized but efficient manner problems of data management, language processor technology and process management for real-time working, that had been tackled in an *ad hoc* manner in existing architectures.

3 Option 7: the synthetic option

The synthetic option, which was to be the basis of the new range, was generated by a small team given the brief of bringing together the best of current thinking on computer design to synthesize an architecture whose only constraint was that of the assumed market environment (i.e. dominated by real-time transaction processing systems, using high-level languages, and large-scale databases). From BLM and MU5 came advanced process management concepts that would dramatically improve the efficiency and robustness of operating systems. (GEORGE 3 and OS/360 had both been notoriously resource consuming and unreliable in their early days.) From sources such as Burroughs and MU5 came concepts of efficient high-level language execution. And from BLM and MU5 came ideas on data management that would enhance the construction and efficiency of database systems.

Without any question, the new range architecture was masterly. It was elegant, efficient, not in the least baroque, and in advance of anything offered by any other manufacturer (and this arguably remains true in the late 1980s)[6].

4 New range: implementation and introduction plans

The output of Phase I of the new range planning activity, which ended in April 1969, was a large body of reports produced by the individual study teams[7]. These reports were examined by, and presentations were made to,

ICL's middle and senior management drawn from across the company. In May 1969, the ICL board officially endorsed the decision to base the new range on the synthetic option. The following month, ICL gave its first public indication of its commitment to a new range by cancelling the 1908A – which was to have been the top of the 1900 series – in favour of a new machine to be known as Project 52 which would be available in 1973.

From mid-1969, the detailed specification of the new range began to take shape. What had actually been produced by the synthetic option team up to this point was a rather slim document describing the broad architectural concepts. To develop a complete specification, the original option team of six was expanded to a full-time team of 12. Team members worked closely with Manchester University and the BLM group to distil their ideas into the new range architecture.

In early 1970, work was started on a marketing plan for the new range. It was intended to support a range of six processors (P0–P5) within the new range architecture, ranging from a low power machine (P0) up to a machine several times the power of the 1908A (P5). In fact, the low power P0 processor was essentially seen as an entry level machine of limited appeal, and the P5 processor (Project 52) was seen as being too expensive to develop without a guaranteed government purchase of at least ten machines, and it was never much more than a paper exercise[8]. Attention was therefore focused on processors P1 to P4, which were viewed essentially in terms of a replacement range for ICL's current computers. For example, P4, the most powerful processor (about ten Atlas power) would be targeted at existing 1906A users and potential 1908A customers. P3 – with about the power of a 1906A – would be the natural upgrade for current 1904A and System 4–70 users. And so on down the range.

The announcement strategy for the 2900 series received a great deal of attention, since there was much accumulated experience in ICL's planning division of the problem of a collapsing order book following the introduction of a new range. The essential problem was that if all the processors in the new range were announced simultaneously, then "the effect on 1900 sales would be catastrophic"[9]. Users of the 1900 and System 4 would naturally cancel their orders until the new range was available, and ICL would have nothing to sell during the two years it might take to switch production from the old range to the new. The ingenious strategy was therefore adopted of a "top down" introduction. The largest processors (P3 and P4) would be announced first, thereby securing the all important 1904A and System 4–70 replacement market. By demonstrating (hopefully) the ease with which users migrated to the 2900 series, and by differential pricing, it was hoped to continue selling the 1901A through 1904A until the P1 and P2 processors became available.

Even though the new range development was now committed to only the P1–P4 processors, funding remained a problem. ICL's total R&D spend, which was projected at about £90 million for the five-year period 1968–73,

had not only to support the new range development, but also the continued enhancement of the current ranges. The negotiation for joint ventures with other companies therefore continued, and were now focused on the French national computing company CII, and the American company CDC. But, as before, these talks came to nothing, although the effect of the negotiations, which entailed many revisions of the new range specifications to accommodate different data standards and interfaces, was to hold back development of new range almost completely during 1970. And during that period the marketing environment itself had changed direction, and the industry was about to fall into recession.

5 Government launching aid for the new range

The two-year period 1970–71 saw the first world-wide recession in the computer industry[10]. The first major casualties of the recession were the giant American company General Electric (GE), which sold its computer interests to Honeywell in 1970; and RCA, which decided to sell its computer interests to Sperry Rand the following year. The demise of GE and RCA as computer manufacturers illustrates the fiercely competitive environment in which ICL found itself in the early 1970s. It had always been realized in ICL that developing the new range would stretch the company to the limit; but the impact of the computer recession on turnover and profits meant that ICL would eventually have to call on the government to provide launching aid in addition to the £13½ million it had agreed to at the time of the merger. In fact, an additional £40 million was eventually provided, but it did not come straightaway and this made the early 1970s one of the most difficult periods of ICL's existence.

In June 1970, the new Conservative Government of Edward Heath was elected. Unfortunately for ICL, the Heath Government had come into office pledged to a disengagement from the direct intervention in industry that had been so much a feature of the out-going Labour administration. This was the famous "lame duck" policy. Consequently, when in early 1971 ICL faced an impending financial crisis, it had to deal with it unaided: during the first half of 1971 over three thousand workers were laid off, and R&D spending had to be reined back. During this period, confidence of users and the City in ICL was badly shaken; and eventually a change in the top management had to be made to restore confidence.

Gradually, however, the political climate began to change in ICL's favour – particularly following the government rescue of Rolls-Royce in February 1971. In early 1971, an inquiry into the British computer industry was conducted by the Select Committee on Science and Technology, and the appearance of its report *The Prospects for the United Kingdom Computer Industry in the 1970's* provided a firm basis for government action. The report was sharply critical of the government's role: "We found it difficult to describe present Government action regarding computer research and development as a policy"[11]. It called for a much higher level of government

support for the computer industry, citing the evidence supplied to the sub-committee by ICL on the much higher levels of support enjoyed by computer companies in the United States, France, Germany and Japan:

> We would not wish to put a figure on the scale of Government funded research and development expenditure which is likely to be required but we anticipate that the sum involved would be not less than ten times the average sum spent by the Government in recent years on computer research and development. We estimate this sum to be not less than £50 million per annum. Any delay in providing money will certainly mean that not only is the objective of independence delayed but also that larger sums will probably be needed to attain the objective at all. We therefore urge prompt action[12].

Even so, the government was reluctant to provide large-scale support for ICL, and instead encouraged it to explore the possibility of merging with an American or European computer manufacturer. These negotiations were to prove very protracted, and in the meantime the government provided ICL with a loan of £14.2 million "to maintain the momentum of its R&D" in July 1972[13]. During the next year a great deal of management time was taken up exploring merger possibilities. In conducting these negotiations, and in accepting the £14.2 million loan, ICL agreed to government guidelines that any partnership it entered into would both retain control of ICL in the UK and would maintain a substantial British R&D capability. In the event, it proved impossible to secure any kind of merger that met these guidelines. By the summer of 1973, the government accepted this position, and agreed to extend its loan to ICL to a total of £40 million. On 4 July 1973, in a statement to the House of Commons, Christopher Chataway, the Minister of Industrial Development, stated the terms of the government's launching aid for ICL's new range:

> The Government have agreed to provide a further £25.8 million in support of the company's research and development programme from October this year until September 1976, making a total of £40 million in all. ... As is normal with launching aid of this type, arrangements have been agreed with the company for the recovery of this £25.8 million, together with the £14.2 million I announced in July last year[14].

The basis for the repayment had, in fact, been the subject of some hard bargaining between ICL and the government. The agreement finally reached was that during the seven year period commencing September 1977, ICL would repay the government a proportion of its profits in excess of an agreed minimum. (In fact, ICL never did make profits exceeding the agreed threshold, and so nothing was ever repaid.)

6 The 2903: a computer for Europe

With funding for the new range assured, the R&D program could press ahead with all speed. But, in fact, while the negotiations with the government

had been taking place, the computer market itself had been undergoing change. This was a change of emphasis away from large centralized mainframes, towards small decentralized computers. ICL's response to this market shift was the 2903 computer; and the 2903 was to provide a large part of the revenue base that would help sustain the new range development[15].

The 2903 project was very much a market-led and profit-oriented one, in which technology was secondary. (Like the decision to launch the 1900 series in 1964, ICL's most commercially successful products have not generally been ones that have stretched technology; and this is no doubt true of most other manufacturers.) The origin of the 2903 computer was a project known internally as PF73, which was ICL's response to the introduction of the IBM System/3 in 1969.

The IBM System/3 computer had been IBM's own response to competitive pressure on its low-end System/360 mainframes in the late 1960s. The adherence of IBM's small machines to the 360 architecture had made them uncompetitive in terms of price/performance, and they were also too sophisticated for small, first-time users. In order to address this problem, IBM launched System/3 in July 1969; the new machine abandoned 360-compatibility altogether and hence could be made to a much lower cost. The IBM System/3 proved an extraordinarily popular machine. Its power and simplicity of use, based on the RPG 2 programming language, combined with low cost, opened up an entirely new market with small businesses that had previously been using visible-record computers or accounting machines. Over 1600 systems were sold in its first year of delivery, and a total of 25 000 systems were eventually sold.

The IBM System/3 had exposed two important new market opportunities, which ICL was not yet effectively exploiting. The first was the very large market of computer-naive small and medium-sized businesses currently using visible-record computers and accounting machines. The second market was for multiple small machines in large decentralized companies. If ICL could develop a small machine, and sell it cheaply in high volume through its new "customer centres" which were then just coming into operation, then it offered the opportunity for the rapid revenue growth, especially in Europe, that was its principal corporate objective: this was project PF73.

The two main product development priorities for PF73 were speed of implementation and low cost. These two requirements ruled out using the small P0 processor of the new range, since that was not scheduled for completion until 1974 at the earliest. There was, however, a micro-coded communications processor under development at Stevenage, known as MICOS 1, which could be rapidly brought into production. It was decided to use this processor to emulate the 1900 series instruction set for PF73, since this would enable software developed for the small 1900 series machines to be used with minimal rewriting; and also to make use of the RPG 2 compiler which was then under development. (The view of some observers that the

2903 was simply a re-engineered 1900 did not appreciate the sophistication of the microcoded processor. ICL had simply elected to use the 1900 series instruction set for economy of software development. If the idea had been more widely applied in the new range, its history might have been much less fraught.) The MICOS 1 was a state-of-the-art processor, and actually a good deal more powerful than was strictly necessary. Again, to maximize speed of implementation and minimize development costs, all the peripherals were based on existing products. These included a disc store of 60 megabytes, which was twice the maximum capacity then being offered on System/3. None of the processors or peripherals required an air-conditioned environment, so that the machine could be housed in any ordinary office (Fig. 1).

Fig. 1 ICL 2903, announced April 1973

In early 1973, the model number 2903 was selected for the new machine. This designation was chosen to give a conscious blurring of the distinction between the old and the new ranges, to create a sense that ICL's new range would not entail an abandoning of the old. This was a subtle and almost self-contradictory objective, but it was achieved. The high 2900s (2950, 2960, ...) were used for the new range, and the low 2900s (2903, 2904, ...) were intended for the small machines. (The numbers below 2903 were not used.)

The ICL 2903 was announced at the Hanover Fair, the main European computer event, on 25 April 1973. The launch was an exceptionally slick

affair: glossy brochures were printed in nine languages, and a film presentation was given in the six principal West European languages. Orders poured in, particularly from Western Europe, and within days the sales target was raised from 1000 to 2000 systems.

7 New range development

With ICL's long-term R&D funding finally assured by the July 1973 government announcement, the new range took centre place in ICL's five year strategic plan for 1973–1978:

> ICL's five year plan essentially revolves around the success or otherwise of the New Range strategy. New Range, in all of its complexity, represents a fantastic challenge to ICL. No major computer manufacturer has announced, and successfully introduced, a new range of computers in the last six years. ICL's five year plan is based on the assumption that ICL will announce, develop, manufacture and install New Range systems with minimum difficulty. In achieving a smooth transition to New Range systems in customer environments, the total organization of ICL will be called upon to meet tremendous challenges[16].

At this time, the new range was planned as a range of processors, from small to large, P0 to P4, with small and large variants of the mid-range P2 processor (P2S and P2L). The concept of a "top-down" introduction conceived in 1970, had been refined and detailed marketing plans evolved. It was planned to announce the top two processors, the 2970 (P3) and the 2980 (P4) in October 1973 with deliveries in mid-1974.

Developing and launching the biggest processors first was both a major technical advantage and disadvantage. The advantage was that by developing the largest processor first, developing the smaller processors later would be relatively easy, since it was essentially a sub-setting process. (By contrast, developing the large 1906 from the 1904 had been a formidable task since it was necessary to enhance an architecture that was optimized for a mid-range machine; IBM had similar problems with its large System/360 processors.) The disadvantage was that all the most difficult development problems had to be tackled immediately. Early software performance would be especially important. The risk was that the entire series would be judged for a long time on the early performance of the new range operating system, and the experience of every manufacturer had shown that a new operating system could take years to settle down. Again, it was imperative that transition aids from the 1900 series and System 4 worked well, or sales of the current ranges would be impacted.

To reduce the risk of an unsuccessful launch, it was decided to postpone the original target date of October 1973 by a full year. On 9 October 1974, the

2900 series was finally announced at an invitation-only press conference at ICL's Putney head office. The launch in fact turned out to be a considerable media event. Because of the government's financial involvement in the new range, the launch was of interest to a general audience, as well as to computer professionals. The Press and TV reportage was wide and almost universally favourable. On 24 October product presentations on the new range were made simultaneously around the world to ICL customers. The 2900 series – "which looked as if it was never coming" – received orders valued at £21 million the same day[17].

The new range launch was, of course, only the end of the beginning. There remained the monumental R&D and manufacturing challenge of actually bringing the machines and software into the field. This task was to consume most of ICL's technical resources during 1975–78. At the time of the 2900 launch in October 1974, only the two largest models – the 2970 (processor P3) and the 2980 (processor P4) – were announced, the first deliveries being made in December 1974 and June 1975 respectively (Fig. 2 and Table 1). It had originally been planned to announce the mid-range 2900 series machines, the 2960 and the 2950 (both based on the P2 processor) in spring 1975, with the 2940 (P1) and the 2930 (P0) following in 1976 and 1977, but none of these deadlines were met.

Fig. 2 ICL 2970, announced October 1974

Table 1 New range processors, 1974–80

Processor	Model	Announced	Delivered
P-series			
P4	2980	Oct 1974	Jun 1975
P3	2970	Oct 1974	Dec 1974
P2L	2960	Mar 1976	Dec 1975*
P2S	2950	Cancelled	
P1	2940	Cancelled	
P0	2930	Cancelled	
S-series			
S4		Cancelled	
S3	2966	Nov 1980	
S2	2956	Nov 1980	
S1	2950	Nov 1977	Jun 1978

Notes
Delivery dates are given for the first four principal processors in the new range, the 2970, 2980, 2960 and 2950. The P-series and S-series used MS1 and LS1 technology respectively. Other models were derived from the S-series processors: from S1, model 2946; from S3, models 2955, 2977, 2988, and others.
*Sic; delivered early to fulfil a government contract.

The development delays were mainly due to the operating system software. During the six year period of the new range development, 2900 software was to consume 35 per cent, or £56 million, of ICL's R&D expenditure, and the operating systems were to account for most of this. This development was the responsibility of the Software Development Organization – which consumed prodigious amounts of computer power for new range software development (Fig. 3). Two major operating systems were under development for the new range: System B for the large processors, and System D for the mid-range machines; these were later renamed VME/B and VME/K. The large-scale operating system VME/B went the way of most major operating systems, and it was released with a low efficiency and a large number of bugs. During this period, VME/B's under-performance started to cause a loss of confidence in the new range. There were some lost orders, and in the case of some installations extra hardware or software maintenance had to be given to get the systems through their acceptance trials and to produce a flow of revenues.

VME/B was a classic evolution-versus-revolution situation. American vendors were supplying mature, stable and reliable operating systems aged seven or eight years, based on older computer architectures though running on the latest processor technology. In moving to a new architecture, ICL had been forced to abandon its own acclaimed and fairly resilient GEORGE 3 operating system. The problems with VME/B had a knock-on effect on the resources available for the VME/K operating system for the mid-range machines. The 2960 and its VME/K operating system were only announced in March 1976, a full year later than anticipated. Even then, VME/K gave less than half of the performance of the equivalent IBM operating system.

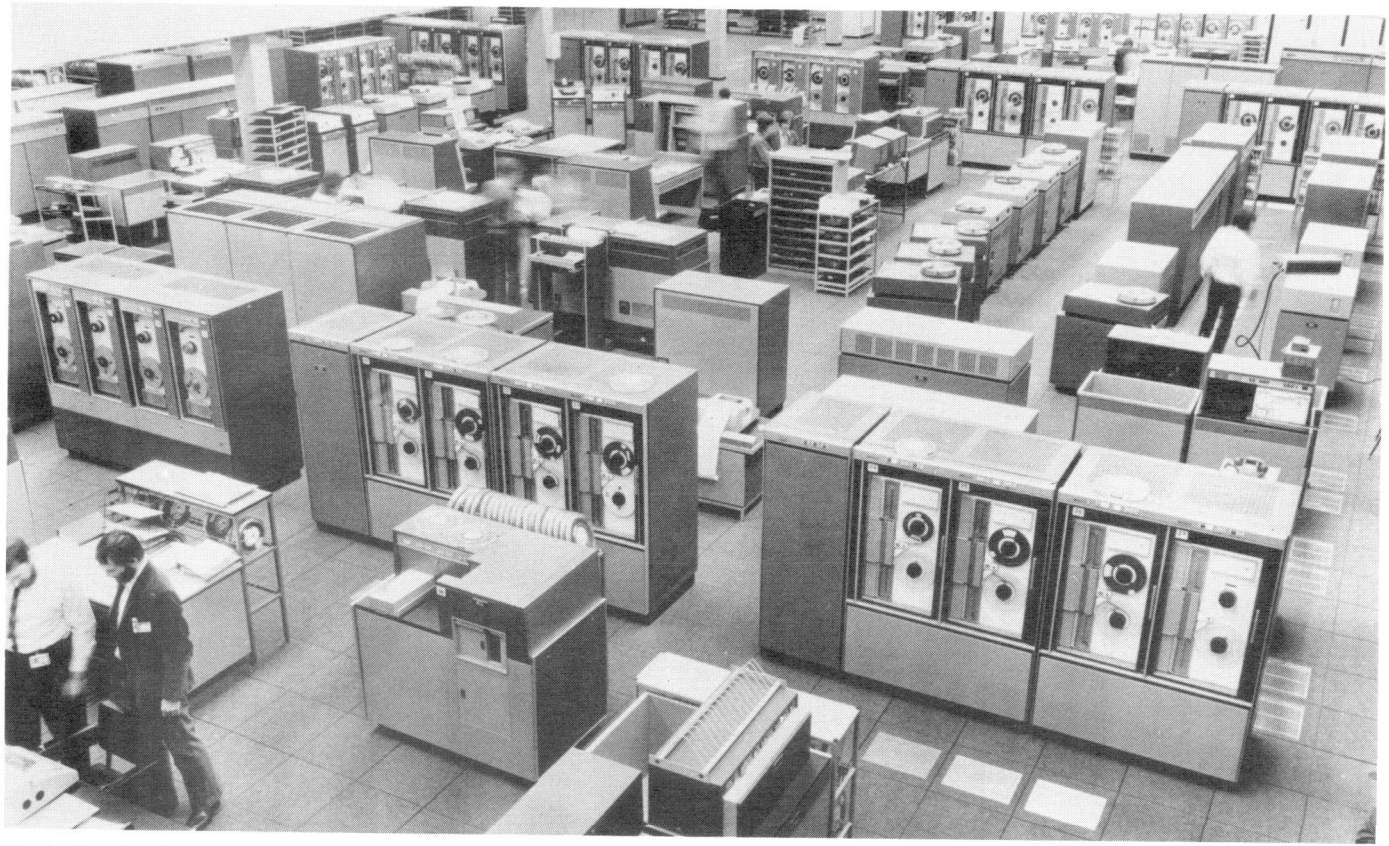


Fig. 3 Bracknell computer hall, 1974

There were also problems with the new range hardware. One of the key problems was the rapid evolution of semiconductor technology from medium-scale integration to large-scale integration throughout the 1970s. In order to keep pace with the change of semiconductor technology, it was decided in late 1975 to develop the S1 and S2 processors in place of P1 and P2; this further delayed deliveries of mid-range machines until 1977 and 1978. As realistic development and manufacturing costs became available for the S-series processors, it became clear that a true range-compatible processor (P0 or S0) for the small 2930 was not economically feasible, and it was cancelled. This left an uncomfortable void between the small 2903 (which was based on the 1900 series architecture), and the mid-range processors of the new range. To bridge the gap, and provide a growth path for 2903 users, an enhanced processor – the 2904 – was developed and announced in May 1976. This was a very necessary market introduction, but coming only two months after the 2960 announcement, it created some market confusion as to whether ICL was introducing a new range top-down, or an old range bottom-up: in fact it was doing both, and the success of the strategy, when the two ranges finally met, would depend on producing good software transition aids.

8 Small business systems and the Singer acquisition

In fact, while ICL's R&D remained focused on the new range, the computer market had continued on its trend towards small decentralized computers. Moreover, as well as the traditional mainframe manufacturers, new competitors had entered the field selling small business systems. These competitors included firms such as Digital Equipment Corporation (DEC), Wang, Hewlett-Packard, and Singer. These new suppliers were all offering low-cost business systems based on minicomputers that came from a different engineering tradition to the mainframe.

IBM was the first of the mainframe suppliers to respond to the changing market for small business systems. Although IBM had had a strong presence since 1969 in the small business computer market through its System/3, this was – like the ICL 2903 – spiritually a mainframe, and very much at the top end of the market. In January 1975, IBM launched its System/32, a much lower cost computer which included a wide variety of applications packages for specific industries, and a basic networking capability which provided distributed, as opposed to merely decentralized, computing for large-scale users. The other mainframe manufacturers quickly followed suit with their new small business systems.

In response to the market pressures, ICL announced a low cost version of the 2903, the model 20, in October 1975. This was, however, not a particularly competitive machine, and there is little doubt that ICL's growth opportunities would have been limited because of its over reliance on the mainframe market, had it not been for what amounted to a piece of good luck. This was the chance to acquire the international operations of Singer Business Machines.

The background of Singer Business Machines (SBM) is as follows[18]. Singer diversified into the business equipment market in the early 1960s, at a time when its sewing machine business was rapidly losing market share due to competition from the Japanese sewing machine industry. With a sales force of 81 000 people in 96 countries, Singer made several acquisitions to broaden its product range and absorb its sales and manufacturing capacity. The most important of these acquisitions was the purchase of Friden Inc. in October 1963, which manufactured low-cost business computers. At that time, only 22 per cent of Friden's business was outside the United States, and the motive behind the Singer acquisition was to use its sales force to bring this proportion up to the order of 50 per cent.

During 1969–73, SBM developed from its Friden acquisition a very innovative and successful range of products. These included the MDTS point-of-sale terminal in 1969, which rapidly became a market leader with over 65 000 units installed. The System Ten computer, announced in April 1970, was the first transaction processing oriented small business system to be introduced – by 1975 some 3500 had been sold, 1400 of them overseas. In 1973, SBM acquired the Cogar Corporation, which added the model 1500 intelligent terminal system to its product portfolio. After this initially successful period, however, SBM began to lose money as the market became more competitive.

In December 1975, Singer announced its intention to sell the business machines division. Immediately following the announcement that SBM was up for sale, talks were held between Singer and potential buyers who included Univac, Honeywell, and Burroughs, and the computer press speculated on many more possibilities. ICL was not, at first, a likely buyer: for while it was enthusiastic about taking on the international division of SBM, it did not want the US business which was considered non-viable – especially for ICL. However, when no buyer for the entire operation materialized, Singer agreed to sell the international division separately.

SBM's three key products, in fact, fitted surprisingly well into ICL's portfolio. The System Ten computer, which was considerably smaller and cheaper than the 2903, would enable ICL to cover the small business system market much more effectively (Fig. 4). It also happened that System Ten had a number of OEM peripherals in common with the 2903, so that the effect on inventory would be minimal. The Singer 1500 series of intelligent terminals, which had an installed base of 4800 units (2900 outside the United States), were superior to ICL's 7500 terminal system, and would reduce its continued development. The Singer MDTS point-of-sale terminal, which had captured 50 per cent of the United States market, was recognized as the market leader; ICL had no equivalent product and it promised to improve an already strong position in retail systems.

In early 1976, there was an intense round of negotiations between ICL and Singer, and on 18 March 1976 an agreement in principle was concluded for ICL to acquire the international division of SBM. Subsequently, ICL also

Fig. 4 ICL/Singer System Ten computer, c. 1976

acquired SBM's Utica, New York, manufacturing plant; this was a conscious decision taken to ensure continuity of supply, and also to internationalize manufacturing operations to make ICL less vulnerable to sterling exchange rate fluctuations and labour unions. The integration of SBM into ICL was achieved in approximately six months. The logistics of integrating the operation into ICL were somewhat daunting, although ICL was perhaps well prepared for it by its long history of merger activity. And unlike the English Electric merger, the products were complementary rather than overlapping. The SBM acquisition effectively doubled ICL's small business systems revenues.

9 New range: declining competitiveness

The mid-1970s saw a quite unprecedented escalation in the pace of innovation in the computer industry – and this escalation took place against a background of declining profit margins. There were two causes of this

technological acceleration and erosion of profits. The first was the arrival of new competitors supplying small business systems; the second was the emergence of manufacturers of Plug-Compatible Mainframes (PCMs). Because of its Singer acquisition, ICL was fairly well placed to withstand the former, but it was in no way prepared for the latter.

The PCM manufacturers began operations in the early 1970s, following the launch of the IBM System/370, when it became clear that IBM was committed to its 360-based architecture for the long term. The first of the PCM manufacturers was the Amdahl Computer Corporation, which began to market plug-compatible replacements for IBM's largest mainframe in 1975. The Amdahl mainframes were developed using Fujitsu semiconductor technology, which at that time was considerably in advance of IBM's. (The relationship between Amdahl and Fujitsu was thus rather like the one that was to develop between ICL and Fujitsu in the 1980s.) The Amdahl PCMs offered a factor of two or better price performance over the corresponding IBM product, and they were an enormous commercial success. During the next two years, several other manufacturers, from the United States and Europe, but particularly from Japan, jumped onto the bandwagon and began to market PCMs. The PCM concept was a turning point for the Japanese computer industry, which for the first time became a significant threat to IBM and the other mainframe manufacturers[19]. In the short term, IBM responded in the only way it could – by cutting prices. This in turn forced the other mainframe manufacturers, including ICL, to lower their prices.

Because of the declining profits from mainframes, the continued enhancement of the new range was to become increasingly difficult. Indeed, by 1977 the new range was already beginning to show several serious competitive weaknesses. These included the non-availability of mid-range processors, the inadequate software transition aids between the old and new ranges, the ageing technology of the existing processors, and the lack of a communications architecture. These major problems were all in addition to the reliability problems of the machines already in the field.

Since the new range launch in October 1974, only the top three members of the series – the 2960, 2970 and 2980 – had been announced. This left a vacuum between ICL's small 2904 and the medium sized 2960 which left the existing 1900 series customer base vulnerable to the competition. In November 1977, ICL finally announced the 2950, based on the S1 processor, and a new small machine, the 2905, was announced at the same time. These new products helped to close the gap, but a mid-range void still remained.

Although the smooth transition from the 1900 series and System 4 to the 2900 series had always been a key objective in the new range strategy, it had been mostly honoured in the breach and ICL was widely criticized for this. Although ICL had actually developed a 1900 series emulator for the 2900

series, it had been restricted to in-house use and had not been marketed, since it preferred to sell the 2900 on its technical merits rather than as a "hot tango 1900". In response to increasing market pressures, however, ICL announced its DME emulators in April 1977, for both 1900 and System 4 users. This stopped some users defecting to the competition and some installations continued to run their 1900 and System 4 programs in emulation mode indefinitely. The move, however, added a further two operating systems to the two that ICL was already supporting.

An important impact of the emergence of the PCM manufacturers had been to make IBM assert its technological leadership, and to shorten the life-cycle of its processor technology, from perhaps four to three years. ICL now considered it was perhaps three years behind IBM on key aspects of processor technology, and that progressive enhancement of all the P-series processors had become urgent. During 1978, plans were laid for replacement of the P3 and P4 processors by LSI processors (S3 and S4), and eventually by full VLSI processors (S3L and S4L) during 1982–84. This evolution would provide price/performance improvements of an order of magnitude over a period of about five years.

Finally, the late 1970s saw a further acceleration of the trend towards distributed computing and computer networks. Responding to the trend, IBM had announced its Systems Network Architecture (SNA) as early as 1972, and the other suppliers were also introducing networks by the mid-1970s. At this point in time, ICL's networking strategy was embryonic to put it at its best.

10 Other R&D projects

Although the new range development program represented the major part of ICL's R&D effort, it was by no means the whole of it. There was also an urgent need to improve ICL's small business systems, and to develop the differentiated products DAP and CAFS.

In the five years following the launch of the ICL 2903 in April 1973, some 2600 systems had been sold, with sales growing at the rate of 39 per cent a year. In spite of this it had never had a corresponding R&D effort invested in it. In 1978, the 2903's age finally caught up with it and sales actually fell 20 per cent over the previous year. This created an urgent need for a successor. During 1978 enhanced versions (the 2903 models 25, 40 and 50) were announced as a short-term response, but in the long term a completely new processor was needed. To achieve a rapid development of the processor, it was arranged to manufacture a little known American design under licence. This was a universal emulator board, known as EMMY (for emulator), created by an American west coast start-up, Palyn Associates. Integration of this bare emulation logic in a full-scale system by ICL design staff led to the ME29 computer launched in March 1980, and which was, in fact, highly successful (Fig. 5).

Fig. 5 ICL ME29, announced March 1980

The System Ten inherited from Singer was also showing signs of age. As it happened, Singer had already developed a successor, the System Ten-220, that ICL was able to use with little development effort, but there was an acute need to introduce cost-effective LSI technology into the processor. Finally, in the small business systems area, enhancements were also needed for the 1500-based and 7500-based systems so that ICL could make an effective entry into the booming office automation market.

The exploitation of DAP and CAFS was to become one of ICL's key product strategies in the late 1970s. One of the consequences of the rise of the PCM manufacturers had been that IBM-compatible mainframes were increasingly becoming a commodity, and the only way of achieving competitive advantage was in terms of price/performance. For the manufacturers of non-IBM-compatible mainframes, product differentiation, which had always been at least as strong a sales argument as price/performance, was now beginning to take on a new importance. Probably the three manufacturers with the best-differentiated mainframe products were ICL, Burroughs and Honeywell, which all had architectures and operating software that were technically superior to IBM. With the most modern architecture and the VME operating systems, ICL was probably the best placed of any of the mainframe manufacturers in this respect.

The falling cost of integrated circuits had also created another opportunity for entry into the computer business by the late 1970s. This was the design of processors with novel, or "non-Von Neumann", computer architectures, which would perform spectacularly well on certain narrow domains of application. In the United States a number of firms began to market products for matrix computation, vision processing, artificial intelligence, and so on[20]. ICL's Advanced Research Group had been working on two such projects for a number of years, DAP and CAFS. By the mid-1970s the time had come to exploit these innovations. A particular attraction of the flexibility of the new range and the VME/B operating system was that special purpose processors – "information engines" as they were coming to be called – could be incorporated into a regular 2900 series mainframe in a straightforward way, enabling users in quite ordinary environments to get a spectacular performance in certain applications.

The Distributed Array Processor (DAP) project had been started at ICL in 1972, with financial support from the government-funded Advanced Computer Technology Project[21]. The DAP consisted of a matrix of processors (initially 32×32) which enabled matrix calculations to proceed at several hundred million computations per second, about three orders of magnitude faster than a large conventional mainframe. The DAP was launched as a product in April 1978. The DAP was highly acclaimed as a technical innovation – it received the BCS/*Computing* magazine award for technological achievement in 1979 – which was excellent for ICL's image, but the relatively narrow market meant that the financial impact on ICL was not great.

The Content-Addressable File Store (CAFS) had the potential for a much wider market. CAFS was actually conceived in 1962, but it was not actively developed until 1969[22]. The CAFS is an ingenious (and heavily patented) mechanism that enables a disc store to be searched "on-the-fly" independently of the main processor. This enables the CAFS to do searching at speeds that otherwise could only be achieved by a colossal mainframe. In 1972 a prototype was completed, and field trials were conducted with the Post Office Directory Inquiries project. The trials were so successful, giving a hundred-fold improvement on conventional computer searching, that in 1975 a CAFS Exploitation Review Committee was established to bring it into the ICL product line. A product, CAFS 800, was announced in 1979. The CAFS products have been showered with technical achievement awards – the British Computer Society Award in 1980, *Computing* magazine Product of the Decade in 1983, and the Queens Award in 1985. All this, however, was very much in the future in 1978; the problem then was paying for CAFS and all the other developments.

11 An approach to the government

In 1978 the central issue facing ICL, as in the past, was in selecting an appropriate level of R&D expenditure and generating sufficient business to

sustain it. The R&D programs underway in 1978 (outlined in the previous two sections) indicated swiftly rising costs, from £36 million a year in 1978 increasing to £89 million in 1983. During the previous five year period, 1973–77, total R&D expenditure had been £123 million, and this was now set to more than double to £266 million in the next five year period, 1978–82. This increase was only partly accounted for by inflation: most of it reflected ICL's broader spectrum of products, such as DAP and CAFS, and its entry into networks and office systems; but the dominant costs were to maintain the competitiveness of the 2900 series, especially the introduction of the LSI processors.

Although in 1978 the funding of ICL's R&D was difficult, it promised to become very much worse in early 1979 when IBM responded to the increasing competition from the PCM manufacturers by making its most dramatic announcement of the decade. On 31 January 1979, it replaced its System/370 mid-range processors with the 4300 series. The new machines offered an unprecedented four-fold price/performance improvement over the machines they replaced. IBM had achieved this leap in price/performance by making a major investment in semiconductor fabrication, putting its technology on a par with the best that Japan could offer. This was the first time that IBM had ever exerted a technological leadership in semiconductors and the impact on the rest of the industry was devastating. Of course the computer industry would have been compelled to respond to the dramatic improvements in semiconductors in the fullness of time, but the effect of the 4300 series announcement was to produce a step-function in the rate of change. For all the mainframe manufacturers the 4300 launch created the competitive environment of 1979. In effect, they were caught in the cross-fire between IBM and the Japanese, and during the early months of 1979 they all cut prices and announced new models.

By the summer of 1979 it was clear that ICL's five-year R&D programme was seriously endangered: profits had fallen, which in turn led to a shortfall in the R&D budget of the order of 10–20 per cent. The major development problem was that the S3 and S4 LSI processors, for the 2970 and 2980 replacements, would be approaching obsolescence by the time that they were delivered. They either had to be expedited or dropped. It was therefore decided to cancel the largest processor, S4, and to allocate the resources to S3 to bring it out as rapidly as possible. At the same time the S3L and S4L programmes were accelerated to bring the VLSI processors out by 1983 or 1984.

ICL was still overspending on R&D, however, and it seemed likely that it would eventually have to call on the government for some financial support[23]. In fact, when government assistance became necessary, it would not prove so straightforward as ICL had supposed. On 3 May 1979 Margaret Thatcher's Conservative Government had been elected into power. Like the Heath Government of 1971, the Thatcher administration was pledged to a disengagement from direct involvement in industry.

During 1980 ICL's position steadily worsened: profit margins continued to be eroded, and the UK was now in the grip of a major economic recession. By the new year, 1981, ICL's internal forecasts were predicting a £25 million loss for the half year, and £50 million for the whole year. It was at this point that it was decided to approach the government for aid. By a rare piece of good luck, ICL's approach to the government coincided with the formation of the new Department of Information Technology; information technology was now high on the political agenda, and the minister Kenneth Baker was to prove very supportive, notwithstanding the government's general policy of non-intervention in industry.

At this time, ICL's major problem was the widening gap between its earnings and the R&D expenditure necessary to keep its products competitive. Whilst previous governments had provided direct R&D support, this was not the route chosen by the new Thatcher administration. Rather, the view was taken that ICL should explore the possibility of merging with another mainframe company in order to achieve a larger market share to fund its R&D. During the early months of 1981, a great deal of management time was taken up in talking to several American companies, including Univac and NCR, as well as Fujitsu in Japan. In fact, the American companies were primarily interested in ICL's customer base rather than in sharing R&D costs, so that a merger would have been unacceptable both politically and from a business viewpoint; this was of course a very similar outcome to the merger talks that ICL had conducted in the early 1970s.

Once the possibilities of a merger had been explored and found unworkable, it was accepted by the government that the solution to ICL's problems would have to be in the form of a cash injection – although the government remained opposed to using its own money. In March 1981, in a meeting between ICL, its bankers, and the government, a highly imaginative solution to ICL's cash problems was put forward in the form of a loan guarantee. Provided ICL's bankers would extend it the £200 million it needed, the government would guarantee the loans against ICL's defaulting. In fact a total of £270 million was provided, of which the government guaranteed £200 million for a period of two years. Since ICL subsequently repaid the loans and the guarantees never had to be called, the assistance given to ICL cost the government not a penny-piece.

In reaching the decision to provide the loan guarantees, the government had concluded – on the advice of management consultants – that ICL's problems were in large part managerial. The loan guarantees were therefore made conditional upon ICL accepting a new management team. On 10 May 1981, the new management team of Christopher Laidlaw (chairman) and Robb Wilmot (managing director) was installed; and they were later joined by Peter Bonfield (now chairman and chief executive of ICL).

Within a few days of taking office in May 1981, the new management began to restructure ICL's affairs, both operationally and in terms of products. The

operational measures taken were all very standard, very unpleasant, but unavoidable if ICL was to become viable again: several plants were closed and workers laid-off, short-time working and early retirement were introduced, and inventories were slashed. The work force cutting continued throughout the ICL recovery, the total headcount reducing from a peak of 33 000 in 1980 to about 20 000 by 1985. The key to ICL's survival, however, lay in its products – and these in turn depended on getting the balance of R&D right. Within six months, ICL's product strategy had been radically reoriented around two themes: mainframe rationalization, and a new "Networked Product Line".

12 Mainframe rationalization: DM/1 and Estriel

Immediately on taking charge of ICL in May 1981, the new management team called for a review of ICL's mainframe products, the unprofitable core of its business. The review disclosed some alarming trends. First, the 2900 series accounted for a disproportionate fraction of ICL's R&D spend: mainframes which produced about one third of turnover, consumed two thirds of overall R&D costs. This R&D burden was inhibiting ICL's participation in the market for small and micro computers, and office systems. Second, market projections showed that mainframe sales were essentially static, and there was no real prospect of greater volume to offset the rising R&D costs. Further, as semiconductor technology moved towards VLSI in the mid-1980s, ICL would not have the volume to justify in-house semiconductor fabrication. Third, the forward plans for the 2900 series were unrealistically ambitious for the market size. In 1981, ICL was supporting five distinct 2900 series processors and two major operating systems. Although there were plans for some degree of rationalization, the mainframe hardware and software commitments were essentially unsupportable – particularly with the shortening product life cycles, and the need for networking software.

The short-term strategy for the 2900 series was therefore aimed at reducing the on-going R&D commitment, and to divert resources to small systems. On the hardware side, the entire 2900 range was reduced to two processors – ME29-based small systems, and S3-based medium-size systems. Both the ME29 and the S3 were, of course, fruits of an earlier period and would not have existed if the R&D momentum had not been sustained in 1979–80. The S3 was launched as the 2966 in June 1981, and derivative versions were launched later in the year. Both the ME29 and the 2966 were major product successes that generated the revenues that sustained ICL during the recovery period. On the software side, the VME/K operating system was dropped entirely in favour of VME/B, which was relaunched as VME 2900 in July 1981.

In the longer term, however, it was chip technology that was at the heart of the problems of the 2900 series. Both the VME architecture and operating system were well proven and competitive, but ICL lacked the semiconductor

technology to manufacture systems price competitive with IBM. In October 1981, ICL – assisted by some behind the scenes activity from the government – succeeded in obtaining an agreement with Fujitsu to obtain access to its semiconductor technology. A key feature of the Fujitsu collaboration was what ICL was to call "technology intercept". ICL would obtain access to Fujitsu's emerging technologies, typically one year before general availability. Fujitsu's technology was considered the best in the world, and certainly better than IBM's – which was what mattered. By designing products based on the best emerging technology, rather than current technology, it was hoped to extend the product life cycle from three to perhaps five years. The technology intercept concept was relatively risky, however, since if the technology did not emerge then neither would the product.

Table 2 ICL–Fujitsu agreement, October 1981

Processor	Mips	Architecture	Design	Software	Manufacture	Technology
DM/1[1]	0·8–2·6	ICL	ICL	ICL	ICL	Fujitsu
Estriel[2]	7–20	ICL	ICL	ICL	Fujitsu	Fujitsu
Atlas 10[3]	15–25	Fujitsu	Fujitsu	Fujitsu	Fujitsu	Fujitsu

Notes
[1] Announced as Series 39 level 30, April 1985.
[2] Announced as Series 39 level 80, April 1985.
[3] Announced as Atlas 10, models 15 and 25, May 1981.

The ICL–Fujitsu agreement fell into three broad areas (Table 2), corresponding to three main product lines: the small DM/1 distributed mainframe, the medium-sized Estriel processor, and the large Atlas 10 IBM-compatible mainframe.* The DM/1 processor would be the replacement for ME29 users and small/medium 2900s. All the architecture, design, software, and manufacturing would remain in Britain, with Fujitsu supplying semiconductor design tools and components. DM/1 was to be based on Fujitsu's state-of-the-art 8000 gate CMOS technology. By exploiting the flexibility of the VME nodal architecture, DM/1 would be capable of multi-processor configurations giving a performance range of 0.8 to 2.6 mips, which was a substantial portion of the lower-mainframe spectrum. The Estriel processor was to be a VLSI replacement for the existing S3 processor. Again, all architecture and design control would remain with ICL, with Fujitsu being responsible for the semiconductor technology. Estriel was to be based on very fast ECL technology using Fujitsu's "top hat" air-cooled technology, which had been proven on its own mainframes (Fig. 6). To minimize production costs and lead times, and to increase Fujitsu's own production volume, the heart of the processor would be manufactured in Japan. Again, by using the VME nodal architecture, Estriel could be configured to give models with a performance in the range of 7 to 20 mips. It was this flexibility of the VME architecture

*DM/1 and Estriel were new names for the S1L and S3L processors. The name Estriel arose because the Japanese had difficulty in getting their tongues around ess-three-el.

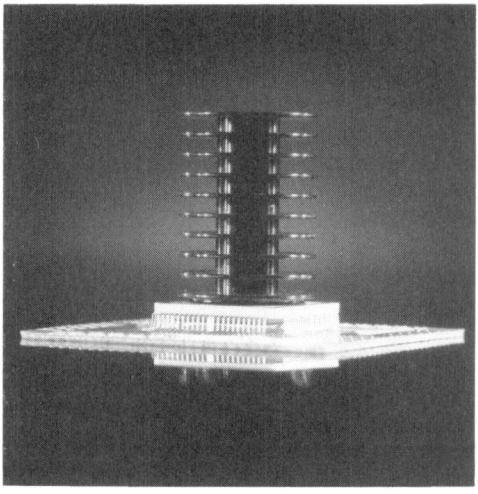


Fig. 6 Fujitsu ECL semiconductor chip used in ICL Series 39 mainframes

that enabled an entire mainframe range to be based on just two processors – a major advantage over ICL's competitors.

The third part of the ICL–Fujitsu agreement was for ICL to market Fujitsu's largest M380 and M382 IBM-compatible mainframes as the ICL Atlas 10 series. This agreement did not really harmonize with ICL's mainframe range, but it was part of the give-and-take between Fujitsu and ICL. It was never anticipated that ICL would sell more than about two dozen machines, but it did offer a "top cover" for ICL's largest users and there was a business opportunity for sales in mixed ICL/IBM sites. As was noticed by press commentators, Atlas 10 left the option open for an eventual move to IBM-compatible mainframes, although this was too far in the future to be an explicit strategy. In fact, the Atlas 10 was a marketing failure and ICL withdrew in 1984. This effectively closed any future likelihood of ICL becoming IBM compatible.

The DM1 and Estriel processors were eventually launched in August 1985 as the first two members (levels 30 and 80 respectively) of Series 39 – the successor to the 2900 series (Fig. 7). The ICL–Fujitsu agreement was perceived as an exceptionally innovative solution to ICL's mainframe challenge, and has since come to be regarded as a classic example of technology transfer in the 1980s[24].

13 NPL: the Networked Product Line

The Networked Product Line was ICL's strategy both to address the technical deficiencies of its product line, and to seize a new marketing opportunity in office systems. The NPL concept is encapsulated in Figure 8,

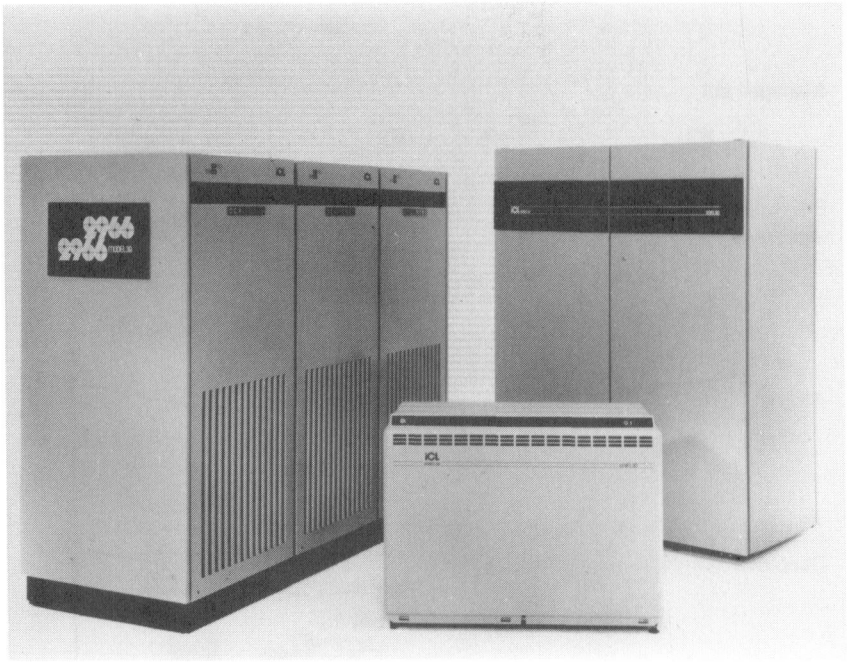


Fig. 7 ICL Series 39, announced April 1985. The use of Fujitsu semiconductor technology brought about a dramatic improvement in performance and reduction in size of VME mainframes. The photograph shows the relative sizes of the Series 39 processors compared with the powerful 2966 introduced in 1981: the level 80 with four times the processing power of the 2966 occupied only half the floor space, while the level 30 with a half of the power was only one-fifth the size. The model 30 used Fujitsu's relatively inexpensive VLSI CMOS technology, while the model 80 used expensive, but much faster, ECL chips

which was first used for publicity purposes in autumn 1981. At that time only a few of the products illustrated had become a reality. An entry into the office systems market had, of course, been a key feature of ICL's product strategy since the late 1970s, but it had been overshadowed by mainframes. The essential change was to shift resources away from mainframes and towards distributed systems based on small and micro computers.

The first product announcements were made in the second half of 1981 (Table 3). In June, System Ten was relaunched as System 25 and provided with networking capabilities so that it could be linked to either ICL or IBM mainframes. In September, the small 7500/1500 computers were relaunched as a fully networked "Distributed Resource System", DRS 20. System 25 and the DRS line were to become major new earners for ICL during the 1980s.

Although rationalization of the mainframe programmes had effectively doubled the resources available for the NPL, it was still necessary both for reasons of development cost and lead-times to make collaborative or

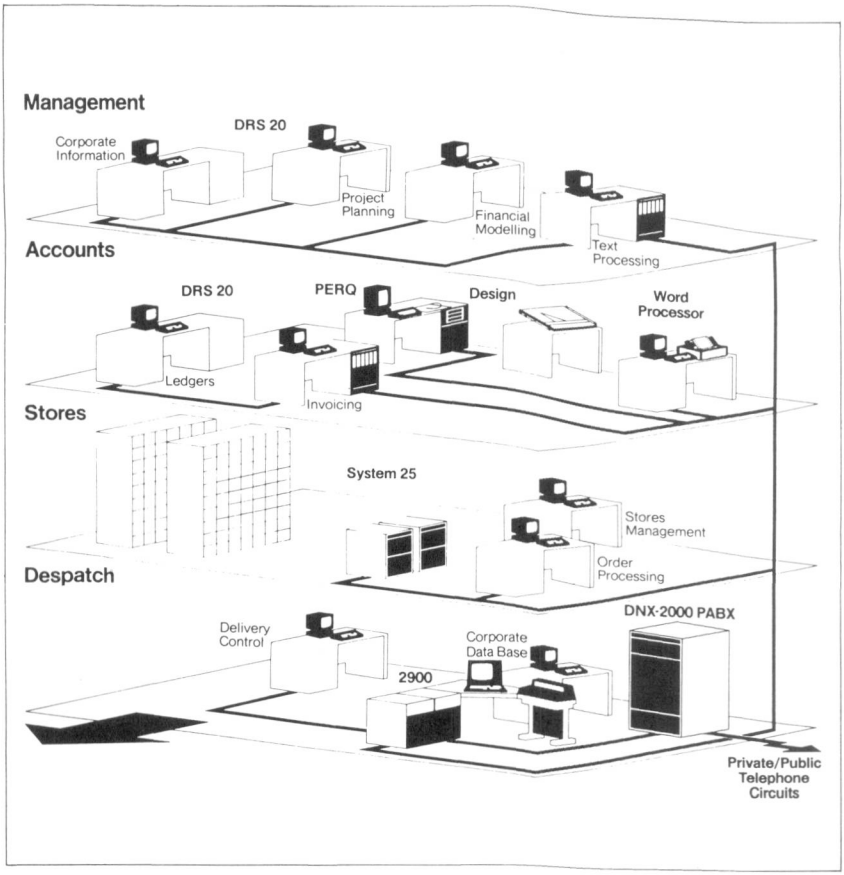


Fig. 8 Networked Product Line, October 1981

Table 3 ICL Networked Product Line

Product	Date announced	Origin
System 25	Jun 1981	Singer System Ten
DRS 20	Sep 1981	Derived from ICL 7500/1500 small computers
PERQ	Sep 1981	Bought/made under licence from Three Rivers Corp, USA
DNX-2000	Oct 1981	Mitel Corp, Canada
PC	Apr 1982	Made under licence from Rair
Wordskil	Apr 1982	ICL 7500/1500, Logica VTS, and Nexos
One-Per-Desk	Nov 1984	Derived from Sinclair (hardware) and Psion (software)

licensing deals to fill out the product range. The first of these collaborations, announced in September 1981, was with the Three Rivers Computer Corporation, an American manufacturer of scientific/engineering work stations. This agreement gave ICL the manufacturing and marketing rights

(excluding the United States and Japan) to the PERQ work station, which at the time was far in advance of anything being made in Britain. A second collaboration was made with the Mitel Corporation of Canada in October 1981 to market its digital telephone exchanges. The DNX 2000 private branch exchange was a major component in the NPL infrastructure, and was a significant step in the convergence of telecommunications and computing. To enter the office automation market, a major need was for specialized wordprocessing software and workstations – these were both acquired through a collaboration with Logica and by acquiring the Nexos office automation company. Again, ICL lacked both the expertise and the development time to make a rapid entry into the personal computer market. An agreement was therefore made with the small UK manufacturer Rair to manufacture its "Black Box" micro computer under licence. The ICL PC was announced in April 1982, which made it a late entrant into the market, although only a year behind IBM. Another innovative product was the One-Per-Desk (OPD), which achieved a convergence of communications and computing in a low-cost, full-function computer/telephone (Fig. 9). To expedite development, the OPD hardware and software were largely derived from the Sinclair QL micro computer (which in turn derived its software from the British software house Psion).

Fig. 9 One-Per-Desk manufacture, 1984

To achieve the networking of the entire ICL product range called for further internal development and external collaboration. The software resources released by the cancellation of VME/K in September 1981 were immediately put to work on the accelerated development of IPA, the Information Processing Architecture for VME mainframes. A key aspect of ICL's

networking strategy was to help establish and implement international "open networking standards" through the OSI standards organization with other, mainly European, manufacturers. This was in sharp distinction to the American manufacturers – such as IBM, DEC and Wang – who all had proprietary network architectures, designed in part to lock-in existing customers. By conforming to international standards, ICL would be able to have its products and terminals "surrounding enemy machines", and would also be able to use the products of other manufacturers to fill the gaps in its own product range. Writing in the late 1980s, the development of OSI IT standards, in which ICL has been a major force, seems to have been one of the most important developments of the decade[25]. The OSI standards will enable European suppliers to compete with the American and Japanese giants, not by the physical merging of companies, but by a loose and informal federation. Individual members of the federation will be able to gain economies of scale by achieving high volumes on a limited number of products which, by adhering to OSI standards, they will be able to integrate with the products of other suppliers.

14 Convergence: the STC takeover

In 1980, ICL and all the other mainframe manufacturers had been faced with two major challenges, one short-term and one medium-term. The short-term problem was to come to terms with the lower profit margins caused by the price-war between IBM and the Japanese plug-compatible mainframe manufacturers. The medium-term problem was to respond to the coming convergence of computers and communications: this convergence implied not merely developing networked computer systems, but also achieving strategic alliances or mergers with telecommunications firms.

During 1981–82, while ICL was tackling its short-term problem of recovery in the marketplace, the medium-term issue of a large-scale convergence had to remain a consideration for the future. Of course, at the product level, convergence was very much to the fore and was implicit in the whole technical and marketing concept of the Networked Product Line. During the recovery period, talks were held at the top level with a number of telecommunications firms with a view to achieving technological convergence and the benefits of greater scale; but nothing materialized – and ICL was in any case then negotiating from a position of weakness. None of the talks, incidentally, involved STC which was to launch a takeover bid in 1984.

By 1983 ICL was seen to have turned the corner; it had returned to profit and was now set for growth. The company embarked upon a £2 million advertising campaign devised by J. Walter Thompson to restore its public image, and to promote itself as "a total systems supplier" through its Networked Product Line, with the slogan "We should be talking to each other". The campaign was memorable and expensive, and marked a turning point for ICL's renaissance in the market place.

On 26 July 1984, entirely out of the blue, a takeover bid was received from STC. The motive for the takeover bid was the coming convergence of telecommunications and computers. The inevitability of this convergence had long been accepted in ICL; consequently, although the STC bid came as a surprise, it was not altogether unwelcome and meshed well with the long-term direction of the information business. After the usual haggling over price, ICL recommended acceptance of the offer and gave a detailed rationale:

> The technology of computers and telecommunications is converging rapidly. Many of the components, manufacturing techniques, research and development programmes and human skills are now shared by both technologies. At the same time, customers are seeking integrated networks of computers and telecommunications equipment.
>
> The merger will combine the strengths of ICL in computer systems and software and of STC in network and transmission systems, thus providing an exciting opportunity to create a group capable of offering a broad range of information technology products and services, including integrated voice and data communication systems.
>
> The combination of STC and ICL will create a strong British group with the resources to meet the challenge of international competition and strongly placed to take advantage of many growth opportunities in the converging computer and telecommunications markets[26].

On Monday 10 September 1984, STC had acquired over 80 per cent of ICL ordinary shares, and the takeover was declared unconditional; ICL now became part of the STC Group, and on 15 April 1985 its name was changed to STC International Computers Limited. As a part of the STC Group, the company was now embarked on one of the most exciting periods in its history, although a short-lived financial crisis in 1985 provided a salutary reminder that in spite of its greater scale, the STC Group remains a relatively small player on the world stage.

Since STC and ICL joined forces in 1984, there have been tangible benefits in terms of operational rationalization and product development. But from the R&D viewpoint the major benefits of convergence lie in the future; and doubtless in ways that cannot be easily foreseen today – for, if one clear lesson emerges from the history of R&D in ICL, it is that the future course of the information business is very difficult to predict.

Acknowledgements

The research for this series of papers in the *ICL Technical Journal* and for my forthcoming book *ICL: A Business and Technical History* would not have been possible without the help of many people. These include some thirty people, from inside ICL and without, who were interviewed in depth; many

people who answered written questions or volunteered information; and several readers in ICL and in the academic community who criticized early written drafts. Full acknowledgements to these people will be made in my book, but I would particularly like to record here my indebtedness to the following ICL people who have supported me during the four years I have been researching and writing the history of the company. David Marwood and Gordon Bates, company secretaries; Gordon Collinson, manager of the ICL Historical Collection; Jack Howlett, editor of the *ICL Technical Journal*; and Arthur Humphreys, managing director 1968–72, and deputy chairman 1972–81.
BOXT

References

This article is based on interviews, documents in the ICL Archives and in the ICL Historical Collection, as well as the open literature. The references below are intended to be representative of the major documentary sources used, rather than an exhaustive listing. Full references will appear in the author's forthcoming book on the history of ICL.

1 CAMPBELL-KELLY, M.: "ICL Company Research and Development, Part 2: Mergers and Mainframes, 1959–68", ICL Technical Journal, Vol. 6, 1988, pp. 171–99.
2 WILSON, H.: "The Labour Government 1964–1970", Weidenfeld and Nicolson and Michael Joseph, London, 1971, p. 8.
3 "Industrial Investment: The Computer Mergers Project", Comnd. 3660, HMSO, London, 1968.
4 IBBET, R.N. and TOPHAM, N.P.: "Architecture of High Performance Computers, Vol. 1: Uniprocessors and Vector Processors", Macmillan, London, 1989.
5 ILIFFE, J.K.: "Basic Machine Principles", Macdonald, London, 1968.
6 BUCKLE, J.K.: "The ICL 2900 Series", Macmillan, London, 1978.
7 Dozens of reports were prepared in the course of new range planning. The two most important summary documents in the ICL Archives are: "Introduction and Summary Documents", Book 1, 30 April 1969; and "Phase II: Situation Report", 20 October 1969.
8 "Project 52", ["strictly confidential, and distribution is on a limited basis"], ICL, May 1971. (J.M.M. Pinkerton Papers.)
9 "ICL New Range Introduction Strategy", 1 December 1971, p. 28. (ICL Archives.)
10 FISHER, F.M., McKIE, J.W. and MANCKE, R.P.: "IBM and the U.S. Data Processing Industry", Praeger, New York, 1983.
11 Select Committee on Science and Technology (Sub-Committee A), Session 1970–71, "The Prospects for the United Kingdom Computer Industry in the 1970's, Vol. 1: Report", HMSO, London, 1971, p. 1x, para. 255.
12 Ibid., p. 1x, para. 258.
13 Hansard, 3 July 1972, col. 34.
14 Hansard, 4 July 1973, col. 530.
15 A detailed history of the ICL 2903 is given in a report entitled "International Computers Limited", April 1976, submitted for the 1976 Award for the Institute of Marketing. (ICL Historical Collection.)
16 "Five Year Strategic Plan, 1973/4 to 1977/8", 30 August 1973, p. 11. (ICL Archives.)
17 "Viewpoint: Great expectations?", Data Processing, November–December 1974, p. 361.
18 FALTERMAYER, E.K.: "Its a Spryer Singer", Fortune, December 1963, pp. 145–8, 154–68. Reprinted as chapter 18 of H.I Ansoff, "Business Strategies: Selected Readings", Penguin, Harmondsworth, 1969.
19 SOBEL, R.: "IBM vs. Japan", Stein and Day, New York, 1986.
20 An excellent exposition of the ICL view of non-Von Neumann architectures is given in M.D. Godfrey, "Innovation in Computational Architecture and Design", ICL Technical Journal, Vol. 5, 1985, pp. 18–31.

21 A brief history of DAP appears in: "ICL Takes World Lead", ICL News, April 1978, pp. 1–2. See also many papers on DAP exploitation in the ICL Technical Journal.
22 CARMICHAEL, J.W.S.: "History of the ICL Content-Addressable File Store (CAFS)", ICL Technical Journal, Vol. 4, 1985, pp. 352–7.
23 The best account of ICL's crisis of the early 1980s is: D.C.L. Marwood, "ICL: Crisis and Swift Recovery", Long Range Planning, Vol. 18, 1985, pp. 10–21.
24 See, for example: M. Pastalos-Fox, "How to Buy Technology", Management Today, December 1983, pp. 78–81. A. Cane, "State of the Art at ICL", Financial Times, 25 April 1985, p. 40.
25 For a discussion on standards see: K. Flamm, "Creating the Computer", Brookings Institution, Washington D.C., 1988, pp. 242–6.
26 "STC Bid for ICL", ICL Press Announcement, 16 August 1984. (ICL Archives.)

Editor's Note

MARTIN CAMPBELL-KELLY'S ICL: *A Business and Technical History* is to be published shortly by Oxford University Press. Publication details will appear at the time of publication in the *STC Gazette* and in the *ICL Technical Journal.*

Notes on the authors

S.J.R. Aitken

Stephen Aitken read Mechanical Engineering at Imperial College, London and gained his initial industrial experience with GEC. He joined ICT in 1967 and worked in sales and support and then in consultancy and business planning. During this period he returned to Imperial College to obtain a Master's degree in Operational Research and Management Science.

Over the last 15 years he has worked in a number of product and industry marketing roles in both Country and Business Divisions, concentrating mainly on applications software. It was during a secondment to Marketing Development and Training that he initiated work on the Marketeers' Workbench. He is now Marketing Manager in STC Networks Industry, which is responsible for developing information systems business with Public Network Operators worldwide.

Dr. H. Alexander

Heather Alexander joined ICL in 1977, having graduated in Computing Science and Mathematics from the University of Stirling. She was involved in database system development until 1984 when she was awarded an ICL educational scholarship at Stirling to undertake software engineering research. During this time she transferred to STC Technology Ltd. and participated in the Alvey-sponsored *me too* project. This work involved collaborative studies with development groups within ICL. At the end of the project she joined British Telecom's System and Software Engineering Centre in Glasgow where she is now manager of a development project.

H. Benker

Hans Benker, born in 1959, has been a researcher at ECRC since 1985 where he has worked on hardware architectures for logic programming systems. He holds a degree in electrical engineering from the University of Erlangen (FRG). His research interests include hardware support for symbolic processing.

Dr. M. Campbell-Kelly

Martin Campbell-Kelly graduated from the University of Manchester in 1968 with a B.Sc. in Computer Science and received a Ph.D. in History of Science in 1980. He is now Lecturer in the Department of Computer Science at the University of Warwick. He is Editor of the Charles Babbage Institute

Reprint Series for the History of Computing and of the *Collected Works of Charles Babbage* and is also an editor of the Annals of the History of Computing. He is presently engaged in a number of computer history projects, and in particular has recently completed a corporate history of ICL, to be published before the end of 1989.

M. Dorochevsky

Michel Dorochevsky, born in 1960, has been a researcher at ECRC since 1987 where he has worked on Prolog compilation and runtime systems. He holds a degree in computer science from the Technical University of Munich (FRG). His research interests include the design and implementation of logic programming systems.

S. Holdsworth

Sean Holdsworth graduated in 1984 with a B.Sc. in Computational Science from the University of St. Andrews. He joined the Computer Science Department of the University of Manchester in 1987 after working as a systems programmer in the National Computer Centre's OSI communications group. At Manchester he has worked in the systems software group of the Alvey Flagship project, and is currently a Research Associate working on kernel software for the Esprit European Declarative System project.

R.W. Jones

Roy Jones is a security consultant in ICL's Defence Technology Centre. He has a degree in Modern Languages from Nottingham University and has worked as a systems designer since 1956. He has been concerned with secure systems and the use of encipherment in computer architecture since 1975, first as a consultant with CCTA and then, since 1977, within ICL. He is a member of the BSI committee IST 20, whose task is to produce standards for the use of encipherment, and of the corresponding international committee ISO/SC 20.

J.A. Keane

John Keane has been a Research Associate in the Department of Computer Science at the University of Manchester since July 1987. He is presently employed on the European Declarative System project, after having worked on the software for a parallel, declarative machine (Flagship) for two years.

He has a B.Sc. in Computation from the University of Manchester (UMIST). His career has included various spells in industry both pre- and post-graduation, including a period with ICL in the VME Director project. Before taking up his present position he was a Research Assistant in the Department of Computation at UMIST working on non-procedural modelling systems.

Dr. K.R. Mayes

Ken Mayes has a B.Sc. and a Ph.D. in Zoology from the University of Nottingham and an M.Sc. in Computation from the University of Manchester (UMIST). He worked for several years in physiological research at the University of Wales College of Medicine, Cardiff, and has spent periods teaching and lecturing in Britain and abroad. He joined the Alvey Flagship project software team at the University of Manchester in 1987, working on the Flagship kernel. He is currently working on the kernel for the Esprit European Declarative System project.

D.G. McVitie

David McVitie joined ICL in 1967 after receiving an M.Sc. from the Department of Computing at the University of Newcastle upon Tyne. He worked on several (then) futuristic projects on database and language integration. In 1970 he left ICL to work as a consultant with Software Sciences on contracts concerning operating systems, languages and databases. Returning to ICL in 1972, he has been involved ever since then with the system design and architecture of the VME operating system and of ICL's Networked Product Line. He is currently concerned with distributed systems and security.

J. Noyé

Jacques Noyé, born in 1959, has been a researcher at ECRC since 1985 where he has worked on instruction set design and compilation for logic programming systems. He holds a degree in engineering from the "Ecole Centrale de Lyon" and a computer science degree from the "Ecole Nationale Superieure de l'Aeronautique et de l'Espace". His research interests include the design and implementation of logic programming systems.

T.A. Parker

Tom Parker graduated from Downing College, Cambridge, with a degree in Mathematics in 1963. He first worked for Ferranti as a technician on a large in-flight simulator for the Bloodhound Mk II missile before joining Fisons in 1966 as Systems Analyst and then Chief Programmer. He joined ICL in 1971 and worked on early design of major 2900 Series operating systems before transferring to Bureau West, where he was responsible for operating system support and for design and development of new security features. He is now a Security Consultant responsible for consultancy on all aspects of technical security within ICL.

He was responsible for the inception and for much of the design of the High Security Option of ICL's VME operating system, and is chairman of ICL's corporate Technical Security Strategy Subcommittee. He is actively involved in the design of high security network architecture within both ICL and standards organisations in Europe. He has written a number of technical papers on security and has lectured both in Europe and in America.

D.J. Saxl

David Saxl joined ICl from the engineering company Rubery Owen in 1973. As a Manufacturing Systems Consultant he was responsible for the implementation of OMAC 29 and other manufacturing systems, both for ICL customers and for ICL's own use within Manufacturing and Logistics (M & L). During 1987 he was manager of the Artificial Intelligence Centre within M & L and is now responsible for the exploiting of M & L's systems and for the development of strategy.

A. Sexton

Alan Sexton, born in 1961, has been a researcher at ECRC since 1987 where he has designed and implemented the operating system of KCM. He holds a degree in physics from Trinity University Dublin. He worked at an applications development company where he helped build a multidimensional decision support system before joining ECRC as systems programmer in 1985. His research interests include operating systems and compiler theory.

Dr. J.C. Syre

Jean-Claude Syre, born in 1948, is the Leader of the Computer Architecture Group of ECRC. He holds a computer science degree from ENSEEIHT Toulouse (1971), and a Ph.D. degree from University of Toulouse (1980). He began research on Data Flow systems in 1971 at ONERA CERT (Toulouse), and built the LAU multiprocessor prototype system. From 1981 to 1984, he was Professor of Computer Science at the "Ecole Nationale Superieure de l'Aeronautique et de l'Espace" in Toulouse. In 1984 he joined ECRC to start and manage research projects in Computer Architectures for Al and Symbolic Processing. He is a member of ACM and IEEE. He was Program Co-chairman of the ESPRIT-sponsored PARLE 89 Conference, and Program Chairman of the 1989 IEEE/ACM International Symposium on Computer Architecture.

Professor S.C. Wheelwright

Steven Wheelwright is Class of 49 Professor of Business Administration at the Harvard Business School. Since receiving a B.S. in mathematics from the University of Utah in 1966 and both an M.B.A. and a Ph.D. in Business from Stanford University in 1970 he has taught at INSEAD (Fontainbleau, France), at Stanford Graduate School of Business and at Harvard. He has also worked for a year as Vice President of Marketing and Sales in a family-owned firm. His most recent teaching assignments have been in the areas of manufacturing strategy, technology, economics, forecasting and planning, and in production/operations management.

Professor Wheelwright's publications include *Dynamic Manufacturing: Creating the Learning Organization*, co-authored with Professors Robert Hayes and Kim Clarke of Harvard University (Free Press, 1988) and *Restoring Our Competitive Edge: Competing Through Manufacturing*, co-

authored with Professor Robert Hayes (John Wiley & Sons, 1984). He has published several articles in the *Harvard Business Review* and in other journals. He is the co-author with Spyros Makridakis of several books on forecasting, including *The Handbook of Forecasting* (2nd edition, Wiley 1987) and *Forecasting Methods in Management* (5th edition, Wiley 1989). His research interests deal with issues that relate a manufacturing functional strategy to the business and corporate strategies. Of particular interest are the management of technology for competitive advantage and the interface between design engineering and manufacturing required for effective new product introduction.

ICL TECHNICAL JOURNAL

Pages contained in each issue

(1) 1–203
(2) 205–405
(3) 407–621
(4) 623–818

Subject index
Volume 6

Author index
Volume 6