



Technical Journal

Issue 1

November 1978

Contents

Foreword	3
Editorial	4
The origins of the 2900 series <i>J.K. Buckle</i>	5
Sizing computer systems and workloads <i>Alan Brock</i>	23
Wind of change <i>G.G. Scarrott</i>	35
Standards for open-network operation <i>Jack Houldsworth</i>	50
Distributed computing in business data processing <i>M. V. Wilkes</i>	66
A general model for integrity control <i>J.B. Brenner</i>	71
ICL Worldwide	90
Notes for authors	95

The ICL Technical Journal is published twice a year by Peter Peregrinus Limited on behalf of International Computers Limited

Editor

J Howlett

ICL House, Putney, London SW15 1SW, England

Editorial Board

J. Howlett (Editor)

D.P. Jenkins

(Royal Signals & Radar Establishment)

M.V. Wilkes FRS

(University of Cambridge)

C.H. Devonald

D.W. Kilby

K.H. Macdonald

B.M. Murphy

J.M. Pinkerton

E.C.P. Portman

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

Annual subscription rate: £5

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

Publisher

Peter Peregrinus Limited

PO Box 8, Southgate House, Stevenage, Herts SG1 1HQ, England

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

© 1978 International Computers Ltd

Printed by A. McLay & Co. Ltd., London and Cardiff

ISSN 0142-1557

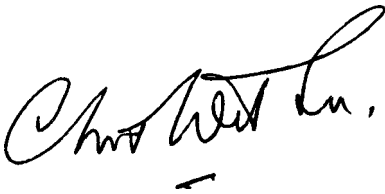
Foreword

It is a pleasure for me to introduce this first issue of the *ICL Technical Journal*.

The electronic digital computer is one of the most important inventions of all time. Its history is very short, scarcely 30 years, but in that time its development has been dramatic, and one of the world's greatest industries has been established. This year marks the tenth anniversary of the formation of ICL; but in the work of the companies from which it was formed ICL has been in this development from the very beginning—I myself and many of my colleagues were personally involved with some of the first computers—and has made some of the most important contributions. This has been possible only because we have always had on our staffs people of great technical excellence and lively imagination, and we must always have people of this kind if we are to continue as one of the leaders in this complex, exacting and fast-changing industry.

Such people are always thinking independently and along novel lines, whether producing new ideas or looking afresh, and often critically, at existing products, practices and concepts. A great amount of this type of thinking, with attendant discussion, is always going on inside ICL, far more than is generally realised. Much of it should be more widely known because it contributes to the knowledge and development of the computational art. Much does, of course, become known through being incorporated in our products and a significant amount is published in the regular technical literature. However, much remains, and it is with this in mind that we are starting this journal. The papers will, in the main, be written by ICL people and will concern the work and views of the individual authors. We have an Editorial Board that includes two distinguished non-ICL members, to ensure that high standards are kept, and it is open to any member of the company to submit a paper for consideration. We shall invite a minority of papers from authors outside the company, and I am very pleased to see that Professor Wilkes has a paper in this first issue.

All the indications are that the journal will bring a great deal of interesting and valuable information to a wide readership. I wish it every success.



Dr Christopher Wilson
Managing Director, ICL

Editorial

The Foreword by the Managing Director of ICL Dr. Christopher Wilson talks about the aims and objectives of this journal. Here I want to say something about the choice of papers generally, and of those in this first issue in particular.

In general, the Editorial Board considers that papers to be published should interest a wide range of people concerned in some way with computers or computing. The Board does not regard the journal as a research journal in the sense of those published by learned or professional societies. In line with this, I shall assume in selecting papers for publication that our readers have a good working knowledge of this very broad subject but may not be familiar with the details given in any particular paper. I shall try therefore to ensure that all papers are reasonably self-contained: in other words, that they do not need more background than the experienced professional computer person can be expected to have.

The Editorial Board felt that the first issue should include papers giving general surveys, to be followed in later issues by more detailed treatments of more specialised areas. This is the style of all those that follow. Two call for special comment: that by J.K. Buckle on the origins of the ICL 2900 range and that by J. Houldsworth on standards in communications. The 2900, as ICL's major range of mainframe machines, is clearly of vital interest to the company and to its customers—and of intrinsic interest as a contribution to computer architecture. John Buckle was a member of the team that produced the basic architectural design and has recently published a book in the Macmillan Computer Science Series. We asked him to write a paper for us based on the book and he has done so. We are grateful to Messrs. Macmillan for permission to reproduce material appearing in the book.

As the use of computers and the services they provide gradually spreads to all members of society, communication with these obviously becomes of greater and greater importance. The practicality of such communication has already been increased by the convergence of the technologies of computing and communications, which are adopting common digital and logic techniques. Much more benefit is yet to come from this convergence and everyone recognises that the specification and acceptance of standards is essential to the realisation of these benefits. Jack Houldsworth of ICL's Letchworth Development Centre is actively concerned with this as a member of several international bodies concerned with standards. He recently wrote a paper for the new journal *Computer Communications* which seemed to us to give an admirable introduction to the subject, and, as with John Buckle, we asked him to write for us. We are grateful again to another publisher, this time the publisher of *Computer Communications*, for permission to reproduce material adapted from that journal.

The length of papers in this issue indicates the range that the Editorial Board has in mind, say 5-10 000 words; but this is not rigid and we shall be happy to publish both shorter and longer papers if the length is appropriate to the coverage of the subject. It is not intended to have a regular correspondence section but I shall be glad to consider written comments on any paper. Also, I shall be grateful for any general comments and for suggestions for topics to be dealt with in future issues.

J. Howlett

The origins of the 2900 series

J.K. Buckle

J K Buckle Computer Consultancy

Abstract

It is several years since the ICL 2900 series was first publicly announced and already the second generation of both hardware and software systems are being installed in customer premises. It therefore seems an appropriate time to record how the 2900 came to be the way it is before the facts become lost in the mists of folklore. Already almost all of the people involved with the architectural design have moved on to other things and the original reasons for particular features or the overall approach are not necessarily obvious to their successors. This paper, then, is an attempt to summarise the initial history of the 'New Range', the aims and objectives that were given to the architectural designers, and some of the fundamental design concepts that were adopted as a result of these constraints. Since the author was concerned with the 2900 from its inception until after its launch the history is complete in the sense that it spans the total incubation period: nevertheless, since the author was obviously not privy to all discussions or decisions that took place, it must be treated as a personal interpretation.

1 Early history

The 2900 series arose directly from the formation of International Computers Ltd. by the merger of ICT and English Electric Computers in 1968. At the formation of ICL it was realised that within a few years the new company would need a range of machines to replace the series currently being marketed by the individual components of the merged corporation. The 1900, System 4 and 4100 ranges inherited by the new company were incompatible in almost every respect, and the cost of maintenance of three production lines pointed to the need for some form of rationalisation.

In addition, although there existed as-yet-unannounced developments of both the 1900s and System 4s that could prolong the life of these ranges for several years all three types of machine dated initially from the early 1960s. Major new design enhancements would be needed to take advantage of improvements in hardware and software technology over the past decade.

As a result of considerations such as these, one of the first corporate decisions of the new company was to establish a New Range Planning Organisation under M.L.N. Forrest. The nucleus of this organisation, known by its initials as NRPO, was formed by 'professional' planning staff drawn from the constituent companies. To this nucleus were added experts in different disciplines from all the operating divisions of ICL: development and manufacturing staff for both hardware and

This paper is based on extracts from the book *The ICL 2900 Series* by J.K. Buckle published by MacMillan Ltd to whom we are grateful for permission to publish the paper.

© 1978 J.K. Buckle

software, logic designers, technology experts, and sales and support staff.

The resulting unit was organised into small teams, each with an individual task. The operation of these teams, who fell into three broad categories, is shown in Fig. 1. (This is an SADTTM diagram.¹ Arrows on the top of boxes represent controls, on the right outputs and on the left inputs. Arrows on the bottom of boxes represent the mechanisms used to carry out the activity of the box. Two-way dotted arrows show interactions.)

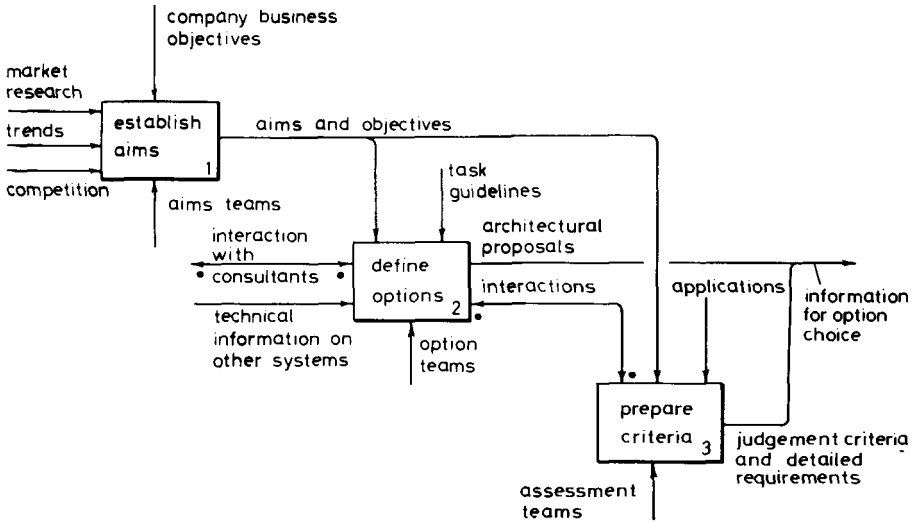


Fig. 1 NRPO operations

The first category was the 'aims' team, who were responsible for establishing the basic aims and objectives of the new range. They carried out market research, determined trends, analysed competitive operations and technological developments, and finally established a corporate marketing strategy.

The second category of team was the 'options' team. Each options team had the task of producing the outline design for an archetypal computer system that could form the basis of a new range. The staff for each options team was drawn from as wide a variety of disciplines as possible—hardware and software designers, technology experts and marketing and planning staff. The teams were kept small to avoid 'committee design', but could call upon the services of a wide variety of experts for advice from within ICL and outside. Each of the teams had a specified 'option' that formed the framework for their design.

The final group consisted of 'assessment' teams. Their task was twofold. First they extended the requirements laid down by the aims teams, by detailed examination of particular aspects of computing with which the new range would be expected to deal effectively. As an example, one of these orthogonal investigations was concerned with data management: the growth of data bases and the growth of techniques for data capture, data transmission and data-independent programming. Another group was concerned with 'bridgeware', the products necessary to ease users' transition from their current systems to the new range. The second task of the assessment teams was to establish detailed criteria by which the output of the options teams could be measured against the stated requirements.

Within the options group, one team considered further, possibly radical, development of the 1900 series to meet the requirements of the 1970s and 1980s. Another team considered a similar proposition for System 4, continuing the RCA and English Electric policy of following IBM. Two of the teams were concerned with the further development of internal research projects that had been in progress for some time. One of these was a dedicated high-level-language machine which had been studied by an ex-ICT team at Bracknell. The other considered the development of J.K. Iliffe's Basic-Language-Machine, ^{2,3} a prototype of which was operational in ICT's Stevenage Laboratories. Smaller teams considered the exploitation of various developments outside the company.

Finally, there was what was called, rather unprepossessingly, the 'synthetic option'. This team had a free hand to consider any known developments or propositions inside or outside ICL and, from any elements it considered good, to synthesise a coherent architectural design. The aim was not under any circumstances to produce a compromise, and the team was at liberty, if it wished, to choose some other option or external development in its entirety. However, its chief objective was to incorporate as many compatible modern, but proven, computer concepts as possible in a unified design. The synthetic option team at this point consisted of six people, including the author. Between them this group brought together a wide spectrum of experience. Their previous work within ICL covered most of the constituent companies, large and small machines from Atlas through to System 4 and 1900, and areas of activity from hardware, logical and technology design, through executive, supervisor, language and application software to corporate planning.

NRPO was not a particularly large unit and it was possible to maintain good communication between the various teams. Thus options teams were able to swap experience and ideas and to take or adapt part of each other's designs. This not only saved effort but allowed the incorporation of good techniques into an option design at an early stage, before design work had become too frozen. Additionally, as work proceeded on refining judgement criteria and developing the options in parallel, it was possible to compare the embryonic designs with the stated ideals. Such co-operation was encouraged and could result in option-design changes to meet criteria or the highlighting of defects in the statement of a particular criterion. Finally, well established communication channels with the operating divisions of ICL, and the availability of the in-house expert consultants mentioned earlier, provided a fast turnaround for new ideas and reactions to them. As a result of this form of organisation, during the first phase of NRPO operations, several of the options were eliminated for marketing or technical reasons, or were amalgamated with other similar options. At the end of the phase the remaining contenders were presented to the company at large and were reviewed by, among others, a 'jury' drawn from senior and middle management across ICL. The outcome of these deliberations was that the synthetic option as defined at that time was chosen as the basis for the future ICL new range. A number of changes and improvements were, however, also proposed. In particular it was suggested that the protecting mechanisms should be improved to give something approaching the capability displayed by the Basic-Language Machine.

Following this decision NRPO entered its second phase and grew in numbers. Refinement and further development of the Synthetic Option proceeded in parallel with an expansion of the basic architectural model to include requirements specifi-

cations for software, initial hardware models, peripheral availability and so on. More experts were drawn in from operating divisions of ICL to help with this work, as well as staff who had previously been associated with other options. By the end of this second phase of NRPO operations the requirements for the initial manifestations of the range were documented to a level at which implementation teams could be established. At this point control passed back to the main hardware- and software-development units of ICL and most of the technical staff returned from NRPO to work on the implementation. The development units began to design and build what were later to be called the first 2900 series systems.

2 Design influences

The very definition of the task of the Synthetic Option team meant that the resulting design would be subject to a wide variety of external influences. In searching for the best modern ideas in computing science and practice, and moulding them into a whole, the team were subjected to various methods and techniques and, even when these features did not appear in the final design, they had an effect on the team's thinking and its approach to solving particular design problems. External ideas were considered and, if attractive, remoulded until they fitted into the partial design edifice that was being built up. This process often introduced much greater generality into 2900 concepts than had been present in the original source. For example, the virtual machine combines a number of existing concepts into a single coherent structure which is new, but the implementation of this new structure is by proven individual techniques.

Many of the direct influences on the 2900 architectural design came, as one might expect, from in-house systems. For example, concepts originating in Atlas and its supervisor and developed on the large 1900s with the GEORGE operating systems⁶ were well known to all the original team and provided a continuous influence on the team's thinking. Again, as already pointed out, there was considerable communication between the rival options. The Basic Language Machine in particular was very carefully studied and many of Iliffe's innovations became foundations for the 2900.

Of the competitive systems studied during the Synthetic Option design, those most relevant were probably the large Burroughs machines. Although the fundamental concepts of the 2900 architecture differ considerably from those of Burroughs, similar design requirements have resulted in an overall design 'shape' that is quite similar. The Burroughs approach of a single high-level-language machine was rejected for the 2900 series but the latter's requirements for good all-round high-level-language performance led to the selection of some similar implementation details. In the operating system area several Multics⁸ concepts had a considerable influence on the 2900, particularly in the sphere of protection where, even eight years on, it still represents the state of the art.

However, the single most important external influence on the 2900 architecture was almost certainly the Manchester University MUS system.^{9, 10} This is hardly surprising. The association between Ferranti Computers, a constituent of ICL, and Manchester University went back to Mark 1 and had progressed through Mercury and Atlas. At the time of the establishment of the New Range Planning Organisation a team at Manchester University had been working on the design of their fifth

machine for some time. Indeed, one of the options briefly considered for the new ICL range was to adopt the MU5 architecture intact. This approach was ruled out since the objectives of the two developments were not identical, but sufficient similarities were present to make the Manchester work of considerable interest to the Synthetic Option team. Two aspects were particularly important. First, both teams recognised the need to generalise the supervisory software system while improving its efficiency, and the decisions taken by the university in this area were well received by the ICL team. Second, both teams were aware of the ever-increasing trend towards use of high-level languages rather than assemblers, and therefore of the need to design a system suitable for the support of the resulting programs. The research and design done at Manchester were of great use in the formulation of the 2900 architecture in this area.

Although there are fundamental differences which mean that MU5 falls outside the 2900 range definition, the early co-operation between the two design teams ensured that there would be a family resemblance between the two systems. Many of the fundamental approaches to storage management and process structure on 2900 are directly derived from the MU5.

3 2900 system features

The 1900 series was conceived from the start as a range of machines that would cover a considerable spectrum of power and facilities. This made it necessary to define the series in terms that were of general application to all the members of the range and not merely to design the first few models that were to be developed. The range definition also had to be system-oriented and to cover items that might be implemented in different ways—purely by hardware, or purely by software, or by some mixture of the two—on different models of the range.

The description of the range in this fashion forms the basic architecture of the 2900 Series; it includes all the most important and interesting features and relationships of the components of a 2900 system. Because of the range and system concepts underlying the 2900, these criteria are not always what one might expect from older range definitions. For example, although all the early members of the 2900 range share the same order code, the order-code definition is not a part of the basic architecture. (In practice, of course, the first implementations are likely to become a *de facto* standard but alternatives are still possible.) On the other hand, considerations of data storage and interchange, as well as international standards, mean that the data formats do form part of the basic architecture: a 2900 is an 8-bit-byte, 32-bit-word machine, for instance. These considerations in turn place restrictions on the features necessary in the instruction set, without actually defining its contents.

The basic architectural description of a 2900 forms a complex hierarchy. At the top is the *architectural model*, which is a general description of a 2900 system. In producing this model, the feasibility of actual implementations was of course considered, but the model was kept as simple as possible, and any specific 2900 system design has to be a practical realisation of this. At the bottom level of the hierarchy are descriptions of implementations of individual hardware and software components which can be combined to form actual 2900 systems. Between the extremes are a number of range and subrange standards.

The actual design of the 2900 did not, of course, proceed in the strict top-down sequence implied above. Even before the Synthetic Option had been chosen as the basis for the new range of machines, detailed investigations of much lower level hardware and software technology matters were proceeding. The results of these had an obvious effect on design decisions at a higher level. Again, at a very early stage the synthetic-option team was forced to sketch out designs for specific systems to convince the members of the team that such implementations within the architecture were in fact feasible. The overall 2900 design thus proceeded in an iterative fashion through the levels, but with the centre of the iteration descending to lower levels as time went on.

4 Aims and objectives

The guidelines for the design and development of the 2900 series specified by the aims and assessment teams took the form of a catalogue of requirements, which were of varying importance and seemed at times to the Synthetic Option team to be too numerous and in some respects self-contradictory. In retrospect, however, it can be seen that the requirements do indeed form a coherent set. Not all affect the architectural model, some being more concerned with implementation details.

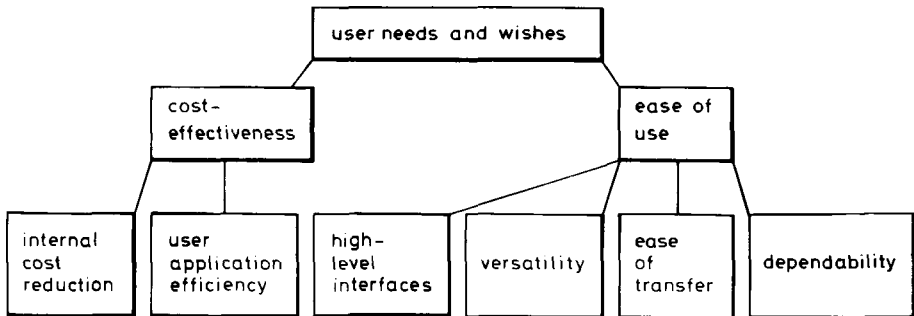


Fig. 2 Design-constraint hierarchy

The requirements can best be considered as a hierarchy that has at its apex the needs and wishes of the user of the system. This hierarchy is set out diagrammatically in Fig. 2. ICL was out to make a system that would be financially successful and it was realised that modern customers were not prepared to take an off-the-shelf system, however good, and adapt it to their needs (or, worse, to adapt their needs to it). The basic requirements for the new range were therefore specified by the way in which its future customers wished to use their computers. The projected customer base was not homogeneous and even within one customer installation the needs of the management, technical staff and end users were liable to be different. However, strong trends could be seen. At the grossest level the users' needs split down into *cost-effectiveness* and *ease of use*. There are two possible approaches to providing a cost-effective system, both of which were taken into account in the 2900 design. The first is to reduce the cost to the customer by reducing ICL internal costs, the second is to make the end systems as efficient as possible in areas of importance to the user.

Ease of use is much more complex, but again broad areas of agreement over a wide user spectrum could be discerned. First, users wanted their dealings with the computer to be at as high a level as efficient use permitted. There was thus a need to define computer systems from the user levels inwards and not from the hardware outwards. A closely connected requirement was that of versatility, both in the variety of applications and in the way in which they were being operated within the system; as batch, transaction processing, remote job entry or multi-access tasks.

Thirdly, it was recognised that the investment in existing software, hardware and training is considerable in most installations and it was necessary to provide protection for this investment. Cheap and efficient methods of using existing hardware, software and data were required. Lastly, users required a dependable system. This led to requirements not only for high reliability but also for the ability to recover quickly and cheaply in the event of failure. These in turn need efficient error detection and recovery methods for both hardware and software.

Orthogonal to all these was the requirement—less explicitly stated but no less important—that the proposed solution be a first-class one from a technical point of view. It was of paramount importance in meeting these requirements that a coherent architecture should emerge. Existing solutions to individual problems need to be combined in such a way as to produce a racehorse rather than a camel.

The way in which these objectives eventually gave rise to the 2900 series is far too complex a story to be recorded here. Because of this, and since the merits of the 2900 series are already adequately discussed in ICL promotional literature, only some of the less obvious points are dealt with in more detail below.

5 Cost effectiveness

The efficiency of computer systems is an elusive entity and difficult to measure. Early measurements of performance, such as addition speeds, have long become meaningless and even less naive work-measurement units are largely irrelevant in the face of very complex user applications. In many business applications the speed of the central processor may be only relevant in terms of its effectiveness at running the operating system. Efficiency to the user is normally concerned with the speed and ease with which his own particular problems can be solved and this is often affected more by data speeds and high-level software effectiveness.

Nevertheless, the speed of basic hardware is by no means totally irrelevant. The basic design of the 2900 series assumed that the early models would use current tried technology but that general design principles must permit the exploitation of possible future developments. In this context it is perhaps interesting to note that after the 2970 had been designed, the hardware-development organisation of ICL was able to build, quite quickly, a number of machines that were logically equivalent to a 2970 but were constructed entirely using old-fashioned (and cheap) 1900 technology and peripheral equipment. These machines, known variously as 'hardware simulators' or 'architectural prototypes', were used to develop engineering and user software before the availability of production machines.

Again, with a range of machines, it was necessary to allow for different mixes of basic hardware components in various range members, and for the use of pipelining and instruction-overlap techniques at the top of the range without prejudicing the performance of smaller 2900 models that could not afford such luxuries. While

most of these requirements affected design levels lower than that described by the architectural model, the Synthetic Option team and its successors had constantly to check that no decisions they took prejudiced the realisation of these objectives.

To be competitive ICL was of course keen to minimise its own costs for both hardware and software development, and manufacturing. The means adopted to achieve this end were basically similar in both hardware and software areas and can be summed up in the two words: *modularity* and *standards*.

As well as the well known and unanswerable technical arguments in favour of the use of modularity in software implementation, ICL had an enormous task to produce a software system that would be competitive. It had to produce, from the outset, a set of software that would be at a similar level, at least in terms of facilities, to the software catalogues on existing machines, which had been built up over several years. It was thus necessary to carry out a complete outline design of the projected software and then implement selectively so that individual components could be improved, replaced or added at a later date without collapsing the whole structure.

In hardware, modularity was again adopted from the outset. However it was necessary to maintain flexible interfaces in the design so that particular 2900 models could exploit new technology as it became available. As a particular example of this, the interface between hardware and software itself was deliberately kept flexible. This has particular importance in the light of the subsequent growth of microprocessor technology.

6 High-level interfaces

By the end of the 1960s, it had come to be realised that the man-machine interface was far too close to the computer. To perform even relatively simple jobs required programmers and operators with a great deal of knowledge and technical skill in the particular hardware and software of the customer's installation. As computer usage increased such people became scarce and expensive. Accordingly there was a tendency to raise the level of the man-machine interface and to make system input as close as possible to the level of the natural statement of the problem or the natural form of the data. This led to the almost universal adoption of high-level programming languages in preference to assembler, the movement towards integrated data-management systems and databases, and in the idea of capturing data where it occurs and delivering results directly to where they are needed.

Such developments eased the problems of scarce staff and long development times but introduced new user worries in terms of efficiency. A machine that has been designed by assembler users for assembler users is unlikely to be well suited to running structured high-level languages. In the data-handling area data-management and communication subsystems tended to be grafted onto existing hardware-software systems. The resulting 'layers', while providing some sort of structured modularity, lead to unnecessary communication problems between the subsystems and to duplication of code and effort in each.

The task set for the 2900 designers was to overcome these difficulties by starting from the outside with the user-interface requirements, and working inwards towards a sympathetic hardware and basic software design. The data-related aspects led to specific requirements on the hardware and supervisory system—the need to

provide at the basic architectural level

- an efficient but totally protected communications environment
- inherent ability to handle a wide range of communication equipment
- fast interrupt handling
- efficient handling of basic data-management functions
- the ability to handle data and code in as independent a fashion as possible
- an effective matching between the design of the hardware and the basic supervisory system.

These requirements had a fundamental effect on the 2900 design contributing to the adoption of virtual machine processing, a hardware stack and descriptor addressing.

The idea of building a machine that was designed specifically for one high-level language was rejected at an early stage because of the large investment in programs written in a wide variety of existing languages throughout the computer community. Investment in all these languages is now so great that often, although new technology that could increase efficiency by a considerable magnitude is available, the cost of replacing existing programs and retraining staff is too large to allow such a revolution to take place. The 2900 designers were thus faced with the more difficult job of producing a computer that would be better than a conventional machine over most languages.

Analysis of the object code and compilation techniques of a wide range of compilers and languages produced a number of desirable attributes for a system architecture that were not language-dependent. Five important requirements emerged.

6.1 A 'clean' order code

To be useful for a compiler writer an order code must be regular in two respects. First, it must be regular with respect to data type; integers, floating point, decimal, single or double length. The hardware should provide the same instruction repertoire for all data types wherever this makes sense. Secondly, order codes normally provide several operand types: operands in store or in registers, operands accessed indirectly via a register or store location, immediate operands contained within the instruction itself. If these are to be effectively exploited by a compiler they should be applicable to all instructions, not just an arbitrary subset, and to all data types. An order code with these two types of regularity, sometimes called an orthogonal instruction set, is a prime requirement for providing simple but efficient compilers.

6.2 Good procedure handling

One tends to associate procedure handling with Algol and its derivatives but the basic subroutine concept pervades all programming languages. Again with the growth of complex operating systems and transaction processing, the interfaces between the controlling system and the high-level-language application program itself are of equal importance and these often show the same characteristics as a proce-

cedure call in one of the more modern high-level languages. Most important of these is that the invoked procedure (or program) requires work space of its own. While it is possible in many cases to assign this work space long before the procedure is invoked, this is not normally an efficient use of resources, particularly in transaction processing or real-time applications, so the basic architecture should handle dynamic assignment and reassignment of storage space for variables, parameters and work space in a simple and efficient manner. This mechanism should be capable of handling the most complex types of procedure and dealing very efficiently with the simpler, more common types. Recursion is necessary in some circumstances and it should also be possible to support code that is re-entrant or shared between different applications simultaneously. Such support must not, however, be allowed to degrade the efficiency of simpler procedure types. Again, with modern programming techniques leading to more procedures per program, it is essential to keep the overheads of the call and exit mechanism down.

6.3 *Good structure handling*

Considered from the point of view of access, program data is essentially of two, and only two, types.

Scalar data items are those that correspond directly to individual units of storage (or small collections of such units). *Structured* data items, on the other hand, are collections of scalar items that are treated in some sense as a named aggregate. While a few languages provide facilities for dealing with these aggregates as entities (APL, for example), in general the individual elements are accessed and manipulated by some form of indexing, slicing or mapping carried out on the named aggregate. With the older scientific languages one tends to think mainly in terms of arrays and with commercial languages in terms of records and character strings. More modern languages have generalised these concepts to provide hierarchical structures that can themselves contain arrays, strings or other structures as sub-structures.

Some special languages contain even more esoteric structures—lists, trees, etc. These, and the need for efficient and convenient access to hierarchic structures, give rise to a new scalar type - reference or pointer variables that can be used to access other variables indirectly. Such scalars can of course themselves appear in structures, giving the possibility for extremely complex addressing patterns.

A 'high-level-language machine' thus needs to assist as much as possible in the access of structured elements, and the detection of access errors. The only efficient and permanent solution to the latter is to provide some form of hardware checking that can be done efficiently by overlapping it with other hardware operations. Such considerations become even more important when the structured data is in some sense self-defining, for example, in database-manipulation languages.

6.4 *Ease of expression evaluation*

Although recent investigation of real programs (see, for example, Reference 11) show that few contain complex arithmetic expressions, it would nevertheless be nice if the 'high-level-language machine' did assist the compiler in this area. Of

much more importance is the need to optimise usage of hardware features in object-program code. For example, although algorithms for register optimisation exist, some general-purpose registers need to be permanently or semipermanently dedicated to specific purposes by the compiler. This wastes the generality of the hardware but the hardware designer of a conventional machine cannot take advantage of such specialisation.

Again, while the hardware designer provides a completely general store, which the compiler then structures into instruction store, temporary data, scalars and data structures, neither hardware designer nor compiler writer can take advantage of the specialisation of data areas. The compiler writer cannot have hardware checks that data or instructions are not being misused and the hardware designer cannot optimise his data-access techniques. Any high-level-language programmer knows that, if he uses the scalar a , there is a high probability that he will use it again soon, whereas if he uses the array element $b[i]$ there is a much lower probability of re-use, but a high probability of an access soon to a different element of b . However a general purpose cache-store designer has no real way of distinguishing between and exploiting these uses.

The hardware designer should therefore have information about the way in which object programs will use his hardware to allow him both to assist such usage and to take advantage of it in his design.

6.5 *An efficient object code support environment*

In thinking of a high-level-language machine one is apt to consider only the efficiency or ease of production of the code that the compiler actually generates. However, the execution of the high-level program will in addition involve code not generated by the compiler directly—library routines and operating system code. To the compiler these items are logically equivalent: they are a method of obtaining a service and can be considered as identical to the call of a high-level-language procedure. In conventional systems, however, the mechanisms are usually very different.

A particular problem occurs with input/output and file handling. Operating systems tend to provide either extremely primitive access mechanisms, or high-level-access procedures, which may mask complex data-management routines but do not meet the requirements of any particular programming language. What is needed is a range of access mechanisms from direct data transfer to sophisticated database handling which can be called in a similar way by the object program, with mapping information possibly being provided separately in the job description for a particular run.

Another important aspect is the way in which the total program—code, data, work space—is mapped on to real storage. If each compiler is forced to take its own decisions on such mapping, particularly with regard to overlaying code or data or both if the total size exceeds the available mainstore, not only will there be unnecessary duplication but we will again forfeit the possibility of direct hardware assistance, and dynamic resource optimisation between programs.

These then are the basic requirements for a good high-level-language engine. Note that not all are direct hardware-design requirements. Some are requirements on basic operating-system design, which in turn may themselves give rise to hardware

requirements. It is worth noting in passing that many of the requirements of high-level-language programs are very important to the production of operating-system software itself. An operating system is, in effect, a large collection of independent and asynchronous processes and has sections that may be obeyed many times a second; in these circumstances, such things as error protection and procedure-entry efficiency become vital concerns. Unless the architectural design helps in these areas the only choice is between relatively efficient but unstructured and bug-ridden systems written in assembly code, or maintainable but inefficient systems written in high-level languages.

7 Fundamental concepts

The remainder of this paper is concerned with the basic principles that underlie the 2900 design; how programs in practice work; how they are conceived by the programmer, transformed into electrical or electronic patterns in the hardware, how execution is then carried out and how this maps back on to the original algorithm. It is in the light of these concepts that the basic architecture of the 2900 series can best be understood.

In examining these aspects of program structure we are concerned with two interrelated ideas. First, how does the program actually use the hardware resources of store, peripherals and processors? Secondly, what is the relationship between the program in the programmer's mind and the actual hardware used and the software that is executed? The second idea itself consists of two parts. We need to study both the way that the final executable representation of the program is structured, and the translation process that takes place from the conceptual to the actual. Consideration of these ideas in as abstract a fashion as possible without the constraints of existing hardware and software leads to some fundamental architectural decisions. These decisions are dealt with in outline in the remaining sections.

8 The virtual machine

The way in which programs use store, processors and peripherals was perhaps first examined systematically in the joint development of Atlas by Manchester University and Ferranti. Essentially we have programs that wish to use more storage or more peripherals than are available to them, or even than there is available on the system as a whole. And conversely we have the fact that no individual program can efficiently use all the available resources all the time.

The second problem was solved by what was originally called 'time-sharing', but is now a generally accepted technique under the name of multiprogramming. Essentially a supervisory system shares the available processor time between several programs in order to maximise the use of other resources - peripherals, file devices, store. Individual programs, however, are unaware of this process and believe that they have the central processor to themselves - a *virtual processor*, in fact. The solution to the peripheral problem appeared long before Atlas, in IBM machines among others. If a program needs two line printers and the configuration has only one we accumulate the data being sent to the two distinct outputs on two print files and then print them sequentially on the single printer. We have created a

virtual peripheral. The Atlas supervisor carried this further by allowing not only more virtual instances of an available device but also the mapping of one device on to a device of a different type.

The degree of virtuality need not stop there, however. Consider access to data on a disc file. The lowest level of physical access is by addressing the disc drive via the controller. At a slightly higher level of abstraction we can address merely the drive, leaving the routing via a controller to be done by some hardware or software 'system'. Higher still we might access a named disc whose drive and controller are discovered by the 'system' but whose volume geometry we know. The next level might be to allow the 'system' to handle volume geometry and deal only with blocks, knowing their physical format and size. Or we could ignore blocking and deal with logical records. Our 'system' has now become some form of data-management system. Finally, we could deal only in named items; we are supported by something like a database system.

Each of these forms of access may be considered as a level of virtuality. Within a total system there will be programs that want to use each of these levels. Each of these degrees of virtuality should therefore be permitted. (For those concerned with efficiency, the existence of these 'layers' on the basic hardware-access mechanism does not necessarily mean that top-level accesses need go through all the intermediate transformations to obtain data. There is no reason why the mapping should not be direct.)

Storage is more complex. Although prices and speeds have changed considerably since Atlas, one can still buy either fast main store or slow cheap(er) store and the price differential has not altered significantly. Programs in any reasonable period of time tend to use only a small part of their total instructions and data and hence, if we can arrange that each such part is brought into fast main store when needed, we can keep the rest on slow cheap backing store. Thus, with some degradation in speed, it is possible to run programs that are larger than the main store available; they have a *virtual store*. Again this mechanism is not even remotely new. Individual programs have used overlay techniques for years. However, program overlaying is costly since the effort of overlaying is duplicated in each program, and, if the system is to multiprogram, overall system efficiency will fall considerably if each program is allowed a partition the size of its maximum overlays.

If, however, we introduce a total-system-overlay mechanism we run into other problems. Variable-length overlays in store will, as they are used and deleted, leave variable-length 'holes' of available space. This will lead to the situation where enough free store may be available to load a needed overlay but it is distributed in unusable pieces throughout the store - store fragmentation, in fact. There is no complete solution to this problem but techniques that alleviate it are known, such as paging and segmentation. The actual techniques adopted in any particular 2900 system design are implementation details but the provision of a virtual store for each program with its own virtual addresses is a fundamental part of the 2900 architecture.

By considering these resource-mapping problems together we can see that the individual solutions that have been adopted for each set of problems actually form part of a coherent pattern. Each program can have a complete *virtual machine* with its own store, addresses, processor, peripherals and files and these will be mapped on to the real available resources by the system. It is this total concept that is perhaps the most important single item in the 2900 architecture, and which allows

many of the other features to be included. While the implementation of the various aspects of the virtual machine may be far from new, the generality of the overall concept allows for some of the 2900 design innovations.

9 The process

The first step in using a computer to solve a problem is for a programmer to find or invent an algorithm. This algorithm is then converted into statements in some machine-translatable language. With knowledge of the requirements of the preceding sections we can assume that this is a high-level language of some sort. It is therefore tolerably close to the programmer's initial conception and this statement is the 'highest' level of expression of the program that we need to consider. This representation is then translated by a compiler into one or more sections of machine-code representation. Some of this translation will be direct in the sense that there will be instructions that correspond in a straightforward manner. In addition some services (such as type-transformation code) that do not correspond directly to the high-level language may be added by the compiler. In other words sections of code have been added to the original statement to make the algorithm work at this lower level of abstraction.

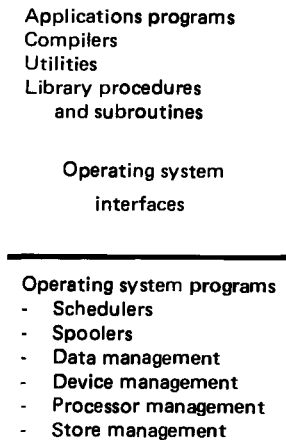


Fig. 3 Two-level model of software

The next stage is to assemble and load the program. Unless the program is very unusual this will result in more code being added to satisfy library calls. However, even this loaded manifestation is not a complete statement of the algorithm in terms of the hardware since the program will during execution also use code in the operating system. Such use may be explicit, as in a 'get-time' command or an I/O initiation, or implicit in the allocation of resources to allow the algorithm to be executed. This conventional software model is shown in Fig. 3. The operating-system code is as much part of the program representation as the square-root or sort routine but it is not 'added' to the program in the same way as the other. Communication between the operating-system code and the rest is slow and ineffi-

cient since it involves crossing the operating-system/object-program boundary. This is to say the least unfortunate, since such communication can be a fundamental part of the execution of the algorithm. Logically, there is no reason why this code should not be added in exactly the same way as library routines and this has many other advantages.

The 2900 architecture therefore provides a *single virtual store* for each program, which contains *all* the code necessary to execute the algorithm. We can call the code for this total representation a *process image*. Although this provides a much cleaner solution than the conventional operating split, two other problems immediately suggest themselves. First we cannot afford a physical copy of the operating system for each process image and we must therefore provide a means of sharing one copy of the code between several processes. This mechanism will not only apply to the operating-system procedures but to library routines and complete utilities. It can allow, for example, multi-access users to share a copy of a compiler. We can also use it to share user-produced code between co-operating programs.

The other advantage of the 2-level model is that software below the line of Fig. 3 is protected from errors in software above it by the needle eye of the operating-system interfaces. This, however, is a very crude mechanism since the single 'fire wall' merely divides the code in a fairly arbitrary manner and does not prevent errors in, say, a spooler, from propagating through the complete operating system without check. And one pays a considerable overhead penalty each time the fire wall is breached. We must therefore also provide some more general and more efficient mechanism for protecting both code and data. Such a mechanism can allow for multiple levels of protection depending on the *privilege* of the code involved. Such a possible multilevel organisation is shown in Fig. 4.

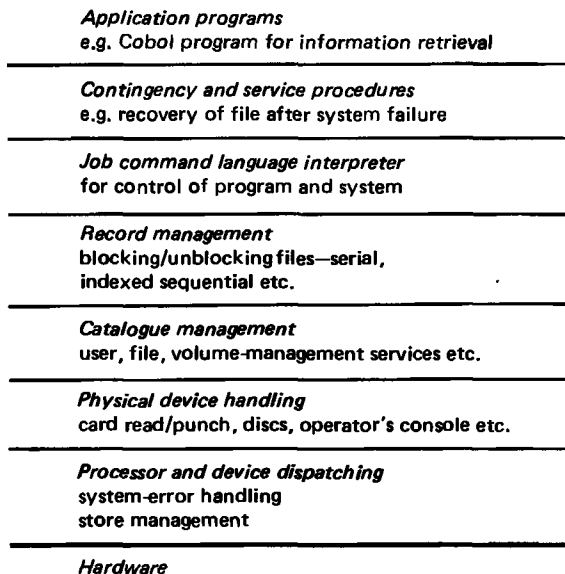


Fig. 4: Multilevel model of software

This provides an additional advantage. Some of the routines (such as conversion procedures) below the operating-system interface barrier of Fig. 4 would be of use to an applications program but are not used because of the high overhead of an operating-system call. Indeed, in general there will be no operating-system call that provides access to them. With a multilevel approach and lower overhead protection mechanism, such facilities can be used, avoiding duplication of storage and effort. This in effect provides the basic mechanism for the user to select the degree of virtuality he requires in peripheral access. The different data-access methods of the last section can be thought of as procedures within the system software, which call on each other for services. Choosing the degree of virtuality is then equivalent to choosing the particular procedure within the chain at which the application program is to enter. Again, because there is now no logical difference between routines in the operating system and in applications libraries, provided that the protection system is designed in the right way, the access to a procedure at any level of privilege can be identical—specifically a standard procedure call. This means that a compiler need have no knowledge of whether an external facility is provided by a library procedure or the operating system. More important even, the compiler need have no knowledge of whether the code it is compiling is to be *part* of the operating system or not. The way in which it invokes other operating-system facilities will be identical to the way in which they are called by an unprivileged applications program. We do not therefore need special compiler versions (or worse, languages) for operating-system development.

10 Program structure

The final set of fundamental concepts arises from consideration of the structure of programs in their high-level-language form and attempting to find ways to provide a system on to which this natural structure can be easily mapped. What then are real programs actually like? Examination shows that most are often badly and illogically structured, but that modern programming techniques are attempting to eliminate this. The basic unit of structure tends to be a subroutine, procedure or *module*; that is, a set of code for carrying out a well defined subset of the algorithm with its own constants, named scalars and data structures. The data structures could be simple things like an individual array or a complex entity such as a Fortran common block.

Fig. 5 shows the structure of a very simple 2-module process. The two modules each have their own code and constants and a set of named items. Some of these latter are scalar variables, others are the names of structures that are represented separately. In addition, there is a need for general work space for use during the process execution. The size of this area will, unlike other areas, vary dynamically as the modules are executed. In the diagram module 1 calls module 2 as a procedure and passes to it a scalar parameter, which is referenced in module 2 as the first of its named items, A2. Note also that the two modules share access to data structure Y. This is common since passing complete data structures as parameters is time-consuming. Sharing scalars is less common but possible, and they can be considered as degenerate structures. In addition, in some block-structured languages at least, it may be possible to access the local names of module 1 from module 2.

What we require in an architecture, then, is a natural support for this form of

program structure and if we are to allow languages such as Algol or PL/1 we must also have system (and preferably hardware) support for recursion in individual procedure modules. Note that this form of logical structure leads naturally to the use of easily shared, pure code since CODE-1 refers to such locations as NAMEA1 and, by providing a new block of names, the same code can be used without interference.

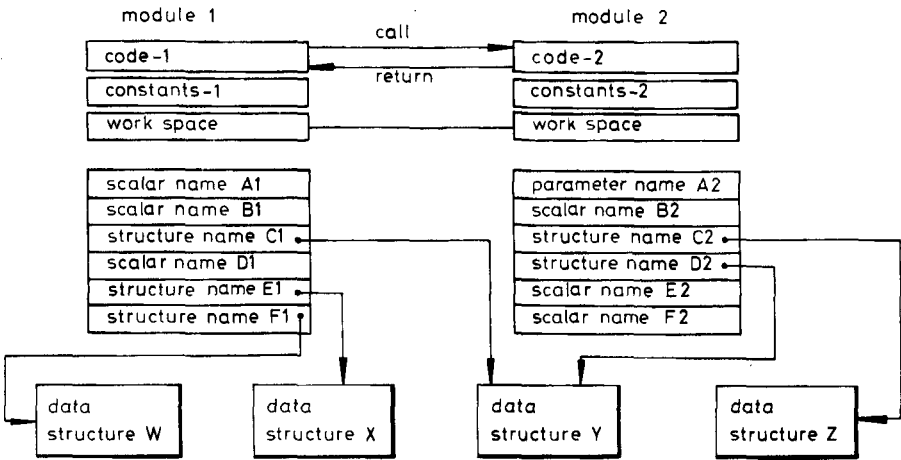


Fig. 5 Logical process structure

11 Basic 2900 architecture

These then are the basic constraints and fundamental ideas that led to the architecture of the 2900 Series. Singly, or in combination, they gave rise to the introduction of a procedural stack, hardware-supported descriptors, virtual-machine and process-support mechanisms, a sophisticated protection system and the other, now well known features that distinguish a 2900 system. Anyone wishing to follow the steps through the formulation of the basic architecture to the first manifestations of the 2900 Series is invited to consult Reference 12.

References

- 1 'An Introduction to SADTTM'. SofTech Inc., Document 9022-78R
- 2 ILIFFE, J.K.: *Basic machine principles*, (MacDonald, London, 1968)
- 3 SCARROTT, G.C. and ILIFFE, J.K.: 'The Basic Language Project' in *Information Processing 68* (North Holland, Amsterdam, 1968), p. 508
- 4 KILBURN, T., PAYNE, R.G. and HOWARTH, D.J.: 'The Atlas Supervisor' in *Programming systems and languages* (McGraw-Hill, New York, 1967) p. 176
- 5 HOWARTH, D.J.: 'A re-appraisal of certain design features of the Atlas 1 supervisory system' in *Operating systems techniques* (Academic Press, London, 1972), p. 371
- 6 'Operating Systems George 3 and George 4'. ICL Technical Publication 4267
- 7 'B6500/B7500 information processing systems' Burrough Corporation, Detroit, 48232
- 8 ORGANICK, E.I.: *The Multics system: an examination of its structure* (MIT Press, Cambridge, USA, 1971)
- 9 KILBURN, T., MORRIS, D., ROHL, J.S. and SUMNER, F.H.: 'A System Design Proposal' in *Information Processing 68*, (North Holland, Amsterdam 1968), p. 491
- 10 CAPON, P.C. and WILSON, I.R.: 'The Compiler Writer's MUS'. Conference on Compilers, Systems and Technology, IERE. Conference Proceedings 25 (London, 1972)
- 11 KNUTH, D.E.: 'An Empirical Study of Fortran Programs', *Software Pract. & Exper.*, 1, 1971, p. 68
- 12 BUCKLE, J.K.: *The ICL 2900 Series* (MacMillan, London, 1978)

Sizing computer systems and workloads

Alan Brock

Consultant Lecturer, ICL Education and Training Centre
Beaumont, Windsor, Berkshire

Abstract

Sizing is concerned with the objective, quantitative evaluation and prediction of computer systems performance. It is therefore a valuable aid to the effective exploitation and development of a computer installation.

1. The general concept

It is self-evident that a data-processing facility represents a substantial investment both in capital cost and in running cost. To achieve a proper return on that investment calls for attention to many things, including the selection of beneficial applications, and staffing and management to develop those applications efficiently. It also requires the efficient utilisation of the computer configuration itself. *Sizing* is the discipline concerned with the quantitative evaluation of this utilisation.

The basic purpose of sizing can be expressed thus:

(a) a given computer configuration has finite resources, expressed in terms of processing power, file accessing etc.

(b) the various jobs to be run on that configuration each demand specific amounts of these resources

(c) sizing is concerned with matching these two.

For any given job running on a given configuration, the demands it makes on the resources can be expressed in terms of:

(a) the time it takes to run

(b) the amount of main store and backing store it occupies

(c) the loadings it imposes on the hardware components of the configuration.

The loading that a job imposes on a hardware component is usually expressed as the percentage of that component's power used by that job. It is often also called the component's utilisation. Thus a job might impose a 40% loading on the processor (CPU or OCP) meaning that the job absorbs 40% of the processor's total power which in turn means that the processor operates for 40% of its time in servicing that job.

The 'size' of a job, expressed in these terms, is important in its own right, indicating as it does the cost of running the job and therefore providing one element in the cost-benefit justification of the job. No less important, the same measures indicate the contribution of that job to the complete workload to be run and in particular allow its effect on multiprogramming to be estimated. Consider for

example two representative jobs to be run on a given configuration. One, job A, is a typical file update with main files on magnetic tape and an amendment file and a report file on a disc; the 'size' of this job can be expressed as component loadings and run time as in Table 1, which indicate that the job imposes only moderate loadings on the configuration and suggest that plenty of power is left for multiprogramming. The second job, job B, is another update but with the main file on disc, updated in situ; its resource demands are as in Table 2. What would the loadings be if these two jobs were multiprogrammed together?

Table 1 'Size' of job A

Component	Loading %
processor	44
input magnetic tape	27
output magnetic tape	27
disc controller 1	25
disc unit 1	57
disc unit 2	3
disc unit 3	-
disc controller 2	9
disc unit 4	-
disc unit 5	9
disc unit 6	-

Run-time, single programming = 42 min

Table 2 'Size' of job B

Component	Loading %
processor	45
magnetic tape 1	-
magnetic tape 2	-
disc controller 1	9
disc unit 1	8
disc unit 2	-
disc unit 3	6
disc controller 2	52
disc unit 4	-
disc unit 5	6
disc unit 6	46

Run-time, single programming = 40 min

The first point to notice is that some hardware components (processor, discs and disc controllers) are shared by both jobs. Consider the effects of sharing the processor. From time to time job A will need to use this; sometimes it will be able to do so without delay, at others the processor will already be occupied by job B and therefore A will have to wait (ignoring for the moment questions of priority scheduling). Similarly job B will, on occasion, have to wait for job A and similar

delays will occur at the other shared components. Thus each will experience delays—queuing delays—and will therefore take longer to run in multiprogramming than if run in isolation. This in turn means that each job will do the *same amount of work* (instructions obeyed, bytes of data read or written) but in a *longer time*: that is, each job will impose a lower loading on each component and therefore the loading on each component in multiprogramming will be less than the sum of the separate loadings of the jobs run in isolation. This is shown in Table 3. These figures show the justification for multiprogramming: although each job takes longer to run, the total loadings and therefore the total amount of work done in a given time are greater than for either job separately.

Table 3 Effects of contention and queuing delays on component loadings and elapsed times when multi-programming

	Single programming			Dual programming		
	job A	job B	sum	job A	job B	combined sum
processor	44	45	89	31	32	63
magnetic tape 1	27	-	27	19	-	19
magnetic tape 2	27	-	27	19	-	19
disc controller 1	25	9	34	18	6	24
disc unit 1	57	8	65	40	6	46
disc unit 2	3	-	3	2	-	2
disc unit 3	-	6	6	-	4	4
disc controller 2	9	52	61	6	36	42
disc unit 4	-	-	0	-	-	0
disc unit 5	9	6	15	6	44	10
disc unit 6	-	46	46	-	32	32
elapsed time (min)	42	40	82*	59	57	59*

*These are the total elapsed times to completion of *both* jobs

Ideally the throughput (the amount of work done in a given time) should increase more or less linearly with the number of jobs in the system. But this could be realised only on an idealised 'infinite' machine that could give every program the service it needed whenever it needed it. On a real finite machine contention for shared components results in queuing delays and, clearly, the contention and delays increase as the number of programs increases. In practice a state is reached at which increasing the number of programs yields little further increase in throughput and may even decrease this. A quantitative understanding of the situation, which depends on the particular configuration and the particular program mix, is clearly important for the efficient scheduling of the jobs. Sizing techniques enable this to be obtained objectively and with acceptable accuracy. Used in this way, sizing can be helpful in many ways:

- (a) assisting in scheduling jobs that do not have deadlines for completion
- (b) for jobs that do have deadlines, indicating which other jobs, if any, can be run concurrently without delaying unacceptably the time-critical job
- (c) explaining quantitatively why certain jobs do not multiprogram together successfully, thus enabling corrective action to be taken

(d) assessing the effects on scheduling and on operational time of new applications still under development, leading in turn to a quantitative assessment of the need or otherwise for enhancement to accommodate the new work.

2 Levels of sizing

Implicit in all the foregoing is the fact that there are two distinct levels of sizing:

(a) application-level sizing, concerned with individual applications or jobs running essentially in isolation, in particular, not contending with other jobs for shared resources

(b) installation-level sizing, concerned with the performance of a complete workload, and taking account of the contention for shared resources.

2.1 Application-level sizing

This can be used for several purposes. It can give a quantitative comparison of the costs, in machine terms, of different design approaches for a given application, e.g. serial versus random. It can reveal the elements of a job that are most costly in terms of machine resources, thus highlighting the elements that will yield most benefit from optimisation. It can form one element in the cost-benefit justification of the application. It can enable the effects of increases in data volumes to be predicted accurately.

2.2 Installation-level sizing

This too has several purposes, notably in scheduling for multiprogramming and for planning the optimum nature and timing of enhancements as described earlier. It can also be used to help in planning file placement when disc units, disc controllers and magnetic-tape controllers are shared by concurrent jobs.

Important though each level is in itself, their full potential is realised when they are used together, essentially in an iterative loop (Fig. 1).

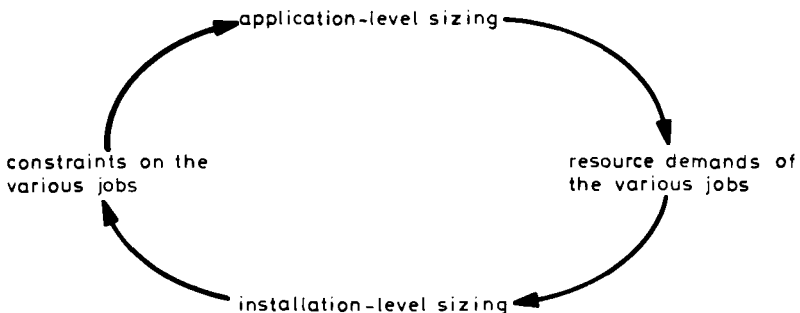


Fig. 1 The two levels of sizing form an iterative loop

The 'sizes' (run times, store occupancies, loadings) of the several jobs are essential input to the installation-level sizing, enabling the benefits of the latter to be achieved. But the installation-level sizing may indicate the advisability of alteration to the individual jobs, perhaps to avoid unacceptable degradation of performance because of overloading of some critical component. For example, there may be two jobs that, because of deadlines, must be run concurrently. If each job is considered in isolation, random processing of disc files may be the optimum solution for both, but that may result in excessive loading of the discs and their controllers, leading to severely degraded performance through queuing delays. Adding more discs and controllers might solve the problem, but it might be much cheaper to redesign one of the jobs for serial processing of magnetic-tape files.

Such compromises and tradeoffs are virtually impossible to assess without the objective, quantitative discipline of sizing.

3 Sizing and tuning

It is important to distinguish between two distinct but related activities: *sizing* and *tuning*.

Sizing can be defined as 'the continuous process of matching a configuration's power to the demands made by the various jobs'.

Tuning can be defined as 'the continuous process of obtaining the optimum throughput from a given configuration with a given work mix'.

Sizing is essentially predictive, helping to answer questions such as:

- (a) What will be the effect, in machine-loading terms, of introducing this new application?
- (b) What would be the effect of adding two more EDS 100s and another DFC?
- (c) What is the optimum job mix between 10.00 am and 11.30 am on Thursdays?
- (d) Can we achieve the required 5s average response time for our new transaction-processing application? If so, how often will the response time exceed 20s?
- (e) How much would the run time be reduced with faster tape units?

Tuning, on the other hand, is essentially corrective. The implicit basic premise is that throughput is inadequate, deadlines are not being met, and it is required to use the existing configuration more effectively in order to rectify the situation. A representative situation is that a workload originally planned for two shifts actually occupies two and a half shifts. Why? What level of multiprogramming must be achieved to get through the work in two shifts? Is that possible, or will it be prevented by overloading of some component? Could the situation be remedied by changing the operating system's installation parameters, and if so, how should they be changed? Would changes in the file placement yield an improvement, by reducing contention for discs and controllers? How much could be gained by optimising the programs? Would tape sorts instead of disc sorts give a worthwhile improvement by reducing the loadings on the disc subsystem?

This set of questions relates essentially to installation-level considerations, but tuning can equally be applied at application level.

The definitions given above for sizing and tuning both include the word

continuous. This emphasises that sizing and tuning ought not to be regarded as 'one-off' activities. The sizing of a new application needs to be done at each major stage during its design, development and implementation and periodically thereafter during its operational life. It is, for example, possible that, during early use, file hit rates may be low enough to make random processing more efficient than serial. After a period of operation the hit rates may increase so much that serial processing becomes the preferable technique. Similarly, installation-level sizing needs to be done initially and reviewed from time to time, to take account of new applications and of trends in data volumes and work load.

4 Data collection

An essential adjunct of sizing and tuning is the collection of data. The data required relates to:

- (a) the current workload
- (b) the projected workload
- (c) the hardware and software characteristics of the configuration.

For the projected workload (i.e. principally the new applications) the quantity and quality of the data will change as each application's design progresses. At the feasibility-study stage, one may have only a rough idea of data volumes, hit rates, amount of processing etc, and many assumptions and 'guesstimates' must be made. The sizing will necessarily be subject to wide tolerances. As design progresses, assumptions can be replaced by firm decisions, estimates can be refined or even replaced by definite values, eg program path lengths. To aid the corresponding refinement of the sizing, all assumptions and estimates must be recorded and, as far as practicable, the sensitivity of the sizing results to errors in the assumptions and estimates should be assessed.

5 Monitoring

Part at least of the data collection relating to the existing workload and the way it runs on the configuration can be done by actual measurement during practical running.

There are three principal sources of data:

- (a) the applications themselves. Many applications record data about their own running, such as file sizes and numbers of transactions processed, often for the purposes of auditing and control. Such statistics can be very useful for sizing and tuning.
- (b) the operating system. Most large, comprehensive operating systems, such as GEORGE 3 and the Virtual Machine Environments for 2900 Series, include data-collection facilities, measuring and logging such things as processor utilisation, peripheral and file-store transfers and paging.
- (c) *ad hoc monitoring*, that is, making measurements specifically for the purpose of sizing and tuning.

Monitoring can be by special hardware or software. For hardware monitoring, a special item of hardware is connected to the configuration being monitored (the 'host machine'). Probes from the hardware monitor are connected to selected

points in the host machine's circuitry. The hardware monitor thus receives signals indicating, for example, when the processor is busy, when the heads on a particular disc unit are moving, when a disc unit is actually transferring data. By careful, planned selection of the signals thus sampled and analysis of them, it is possible to derive an accurate and informative picture of the use of the hardware, including not only the loadings on the components but also such important details as the amount of overlap between processing and input/output, or how the processing is split between object programs and operating system. Most software monitoring is in fact carried out by the facilities intrinsic to the operating systems as already mentioned, but other facilities such as program-trace routines are also used. By tracing the actual execution of a program, these enable actual path lengths to be measured and the highest-cost sections of a program to be identified, these latter being, of course, the prime candidates for optimisation.

Hardware and software monitoring are complementary rather than alternatives, since each yields its own type of data organised on a particular basis. For example, consider processor utilisation. A hardware monitor will normally measure the processor utilisation at regular, frequent intervals during the test period, producing, in effect, a graph of processor loading against time. It will not normally give a breakdown between concurrent programs. Software monitoring, by contrast will usually indicate how much processing was done for each program, but not how that was distributed over the test period. A hardware monitor can show how often the heads moved on a particular disc unit and how much time was spent in head movement, but not which programs caused those head movements. Software monitoring can show how many disc accesses a program caused, but not how many of those accesses caused head movement nor how much time was thus consumed.

Thus a complete picture of how the configuration was used, moment by moment and by the various jobs, really needs the co-ordinated use of hardware and software monitoring.

6 Sizing techniques

The range of techniques available for sizing studies is outlined below. The principal groups are:

- (a) gross sizing
- (b) modelling
- (c) analytical methods
- (d) experiment

6.1 Gross sizing

This uses simple, quick methods to obtain approximate answers. It is based on relatively crude measures such as 'I estimate that this job is equivalent to 500,000 Post Office work units (POWUs) of processing and involves 20,000 random accesses to a disc. The processor power is equivalent to 400 POWU per second, and an average disc access takes about 120 ms. Therefore, the job will take 1250 s for processing and 2400 s for the disc accesses. Assuming an average concurrency of 1.5, then the run time will be about $(1250 + 2400) \div 1.5$ s i.e. 2400 s or 40 min. The loadings

will then be about 1250/2400, say 50% on the processor and 2400/2400 = 100% on the discs'

For some purposes, gross sizing is good enough. The illustration above indicates that the run time will be in the region of 40 min, not 10 min and not 4 h, and that the discs will be very heavily loaded. That may be all one needs to know. Even if such crude answers are not accurate enough for the purpose of the sizing study, they serve as a valuable check, helping to reveal any mistakes that may be made when using more accurate, but more complicated, techniques.

6.2 Modelling

Modelling embraces two principal methods: barcharting and simulation. A bar chart is a diagrammatic representation of the activities in a program. A line is drawn for each hardware unit used by the program, and on that line are marked the periods when that unit is actually functioning. The bar chart in Fig. 2 is part of the complete chart for the case of an update program with main files double-buffered on magnetic tape in which, for each block of the main file, there must be a random access to a reference file on disc. The relatively infrequent accesses to the file of input transactions and to the print file are ignored. It will be seen that the bar chart reveals a regular cycle, from the start of processing one block to the start of processing the next. Once that cycle has been discovered from the bar chart the required results follow easily. The run time is simply the duration of that cycle multiplied by the number of blocks in the main file. The loading of each component is given by the percentage of the cycle time for which the component is active.

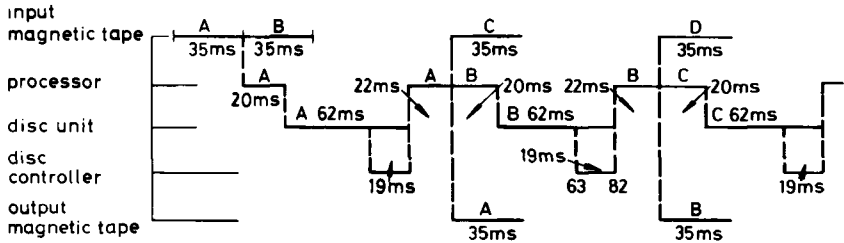


Fig. 2 Sample bar chart

Given: Time for magnetic tape block = 35 ms; Disc time per access = 62 ms; Disc controller time per access = 19 ms per block; Processing time per block = 20 ms before disc access and 22ms after

Then, from barchart:

- elapsed time per block = 20 + 62 + 22 ms = 104ms
- magnetic tape loading = 35/104, i.e. 33.7% each
- processor loading = 42/104, i.e., 40.4%
- disc unit loading - 62/104, i.e. 59.6%
- disc controller loading = 19/104, i.e. 18.3%.

Except for first and last blocks, magnetic-tape accesses completely overlapped with other activities. Accessing disc file accounts for 59.6% of the elapsed time.

A bar chart is also valuable in providing a clear picture of how the program uses the hardware. Fig. 2, for example, shows clearly the large proportion of the cycle time that is due to accessing the reference file, suggesting that thought could well

be given to reducing the access time, or even to eliminating the need to access the file at all.

Simulation is usually by special simulation programs. They normally yield very accurate results and can deal with complex cases. Usually, though, considerable effort is needed to learn how to use one effectively.

6.3 Analytical methods

These involve the use of models expressed in mathematical terms.

Sometimes such models are adaptations of the bar-charting procedures described earlier, but expressed in a more formal, mathematical way and using statistical techniques to represent the distribution of lengths of a bar in the chart.

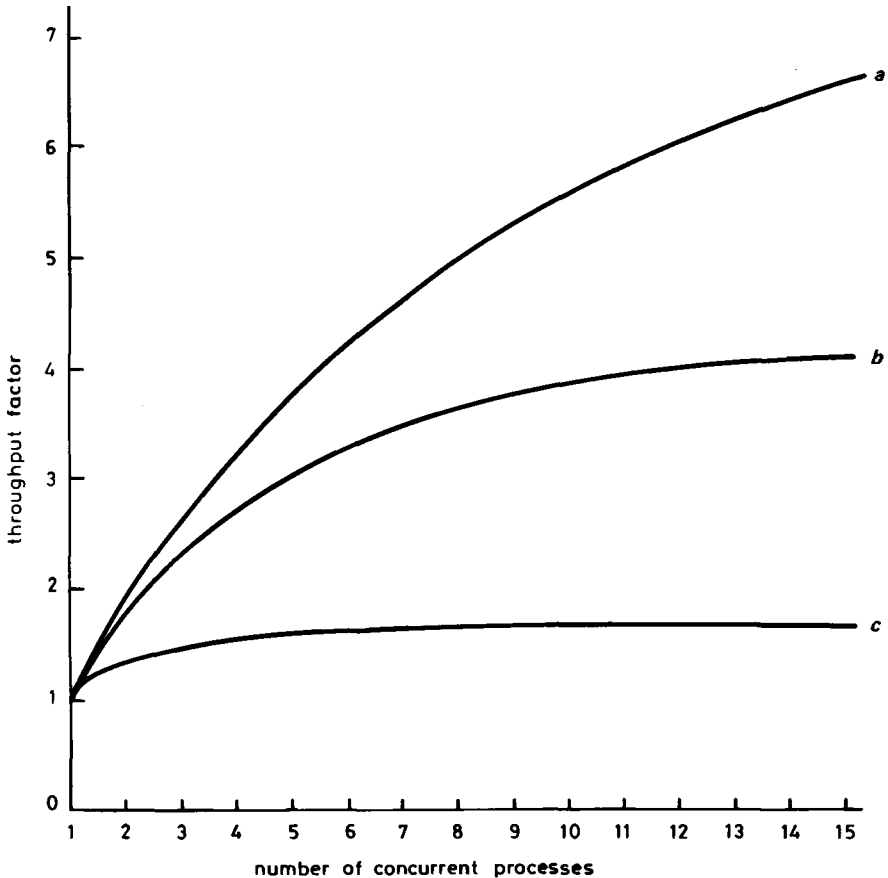


Fig. 3 Three sample FAST curves

- (a) Evenly loaded configuration
- (b) Relative loadings corresponding to last column of Table 3
- (c) Configuration with one component much more heavily loaded than the others.

However in most cases such models are based on queuing theory. Simple queuing theory enables one to predict for example the average time one has to wait for a ticket at the station in terms of the rate customers arrive and the rate at which the booking clerk can serve them.

Simple queuing theory can be very useful for calculating delays at particular points in a system where there are a large number of customers, such as messages in a transaction system. However, in many cases the total number of customers is quite small, in which case more comprehensive variants of the theory can be used.

A more general form of queuing theory deals with the movement of a finite number of customers in a network of servers. Models of this type have been in use since the early 1970s and are known in ICL as FAST (football analogy for system throughput). The idea here is that the operation of a batch system can be likened to a bunch of kids kicking a football around. The kids are system components such as the c.p.u. and the peripherals. The football is a program.

If now they are given several balls to kick around, their total throughput in kicks per second will increase. The theory enables this increase in throughput to be calculated, given the amount of work each player (alias server, alias system component) has to do and the number of footballs (alias programs or processes) in play.

FAST can be used, *inter alia*

(a) to estimate the effects of enhancements to a given configuration (and hence their cost effectiveness)

(b) to estimate the effect on throughput of a change in the workload.

(c) to estimate the effect of changes in the way in which the workload is run as a result of relocating files in the drum/disc/tape hierarchy.

Fig. 3 gives some typical curves resulting from applications of FAST.

6.4 Experimental methods

Experimental methods are in principle the most accurate and reliable methods for sizing. Perhaps the best known is benchmarking, in which a job or a mix of jobs is run on an appropriate configuration and the performance measured. There is, of course, the problem of obtaining time on a suitable machine. More significant, perhaps, is the difficulty of ensuring comparable standards of operating and tuning between the benchmark and the everyday, production running of the job. Also, the jobs in a benchmark are often 'typical' jobs rather than 'actual' jobs, introducing a further degree of uncertainty into any conclusions that may be drawn.

Another experimental technique is the so-called 'network simulator' intended for testing and measuring the performance of online systems (TP or MAC). To test a TP or MAC system in real operation necessitates having the mainframe computer, the network, N terminals, N trained terminal operators and N observers to monitor the operators, and to have them all available for a period long enough to give a realistic test. Assembling those resources may not be easy. A network simulator is a program that generates messages as though from terminals and loads them at a controllable rate directly onto the mainframe machine in the absence of the communications network etc.

Since the mainframe and its software are being exercised under (reasonably) normal operating conditions, the results are, in principle, very accurate. However, if the network simulator program runs in the mainframe itself, then it absorbs some

of the power of the mainframe and thereby interferes with the very thing one is trying to measure. To overcome this, the simulator program should ideally run in a separate machine with a direct data link to the mainframe under test. That too may be logistically difficult to achieve.

Table 4 shows the applicability of the techniques outlined above to the different areas of sizing studies.

Table 4 Applicability of sizing techniques

	Gross sizing	Bar charts	Simulation	Queuing theory	FAST	Benchmarks	Network simulator
Application level batch	J	J	J	X	X	J	X
Application-level communications	J	X	J	J	X	J	J
Installation-level batch	J	?*	J	?†	J	J	X
Installation-level communications	J	X	J	J	?‡	J	J

Notes: *Not practicable for more than two simple programs
 †Provided interactions are limited and simple
 ‡Communications-based applications treated effectively as batch jobs over periods short enough for communications load to be regarded as constant.

7 Uses and benefits of sizing and tuning

The principal uses and benefits of sizing and tuning have been suggested in the foregoing discussion. Essentially, they can be expressed as the use of objective, quantitative measures of the current and future workload and of the capacity of the configuration, to provide a basis for decisions relating to the effective exploitation and development of a computer configuration.

It is important to realise that sizing and monitoring are tools to aid decision making. Therefore every sizing activity should have explicit objectives—essentially, a question that can best be answered by a sizing study and the answer to which can lead to beneficial action. Therefore the effective use of sizing requires a perception of the type of question that sizing studies can answer, together with the ability to interpret the results and use them as a basis for decision and appropriate action. In that, sizing is no different in kind from many other activities in data processing, and in commerce and industry generally.

The paper has given examples of the types of question that can be answered by sizing studies. Anyone whose responsibilities require the answering of such questions could well consider sizing as a helpful tool.

References

1. BERNERS-LEE, C.M.: 'Four Years Experience with a Performance Methodology for Systems Planning', Proceedings of European Conference on Computer Performance Evaluation, 1976, ONLINE, Cleveland Road, Uxbridge, Middx.
2. BERNERS-LEE, C.M.: 'Understanding System Behaviour through System and Work-Load Modelling', Infotech State-of-the-Art Report on System Tuning, 1977, Infotech International Ltd., Maidenhead, Berks.

Wind of change

G.G. Scarrott

Manager, ICL Research & Advanced Development Centre,
Fairview Road, Stevenage, Herts

Abstract

The paper outlines the analytical arguments underlying the selection of the projects undertaken at the ICL Research & Advanced Development Centre. The arguments are derived from consideration of the potential scope of the business of ICL regarded as an information-engineering company. Four of the recent ICL projects are described in outline: Variable Computer System; Content-Addressable File Store; Distributed-Array Processor; and man/machine interaction by speech.

1 Introduction

The ICL Computer Users' Association took for its 1978 conference* the theme 'The wind of change'. It is, therefore, appropriate that we from the Research & Advanced Development Centre (RADC) should discuss this theme, since it is our responsibility to be aware of the 'winds of change' and to ensure that the Company is prepared for them. I should emphasise that industrial research departments such as ours do not blow these winds: it is the worldwide community of users and suppliers of information systems who collectively determine events and decide when change shall be inhibited, which changes shall be encouraged and when change shall become a hurricane. It is the role of industrial research to forecast the inevitable and prepare for it: in a word, to be the barometer and not the tempest.

I shall first discuss the role of ICL Research and our point of view. This leads naturally to a consideration of techniques for technological forecasting, a ten year forecast for information engineering, some consequential ICL research projects and conclusions for the Company and its customers. I have used the term 'information engineering', which I feel best describes the field covered by ICL research; perhaps an unfamiliar term but an appropriate description of the computer business now that it has become of age. Let us look into the precise meaning. According to Webster, an 'engineer' is an ingenious fellow who puts scientific knowledge to practical use—indeed the very word 'engineer' is simply derived from 'ingenious'. Thus an 'information engineer' is concerned with the deployment of available technology to meet the information-handling needs of society. I suggest that each of us in this industry could legitimately cherish an honourable ambition to be such an engineer by cultivating a deep understanding of the social purpose of our wares as well as skills in designing and making use of them.

*This paper is based on a presentation made to the ICL Computer User's Association at their Annual Convention, 1978

2. Point of view

To undertake this task of industrial research we must be keenly aware of the present situation in our industry. In order to concentrate our attention on essential issues we must try to distinguish between 'what' purpose an information system must serve and 'how' it operates to serve such a purpose. Jargon in the computer business constantly proliferates and serves a useful purpose as a shorthand for describing how things work, but the very usefulness of jargon tends to obscure what it is all for, so that we deliberately try in our own discussions to minimise the use of jargon e.g. 'operating system' and 'compiler' are both dangerous words since they imply what is done at run time, which is strictly an irrelevant detail. Another consequence of our insistence on understanding the ultimate purpose is that we tend to regard established interdisciplinary boundaries, such as that between hardware and software, to be negotiable. Indeed, all boundaries must be regarded as negotiable with the exception of only two.

One is the boundary between the information system and its ultimate user and the other is between the information system and the suppliers of physical devices, since the components must be potentially available. In a word, only the physical and human realities are not negotiable.

3 Systematic forecasting

We must first consider which way the wind of change is blowing and which apparently well established practices are likely to be swept away, so that we are prepared to ride the events when they occur. To do this, we must endeavour to forecast events well before they occur. Such forecasting is hazardous in any business, but particularly so in the computer business, with its feverish rate of technological innovation and incomprehensible jargon. It is possible to minimise such hazards but only by cultivating a sense of history—a feel for the technological evolutionary process. We can never achieve certainty but forecasting based on an analysis of the historical record is certainly more trustworthy than science fiction over the period of about ten years for which systematic forecasting is possible and necessary. A longer term forecast is of somewhat academic interest, since, even if we got it right, few would believe it and fewer still would regard such a forecast as a spur to action. I propose first to take a broad view of the general technological evolutionary process and then concentrate on the expected events in our particular field of information engineering.

Let us consider the technological evolution of a typical product in the four phases: birth, adolescence, maturity and senility. When the new product first appears, the technology for making it is primitive, but, if it serves a useful purpose at all, pioneering users exist whose needs for the new products are so pressing that they are willing to adapt their practices to take advantage of it. Thus, in the beginning most of the discussion is concerned with 'how' to make and use the product and there is very little discussion of 'what' purpose the product should serve or what should be its technical specification to serve such a purpose. In the adolescent stage the main population of users begin to appear. Many of these do not have such a clearly recognised need, and, indeed, some of them may be only following fashion. As a result of this, discussions regarding the new product begin to tackle the more

fundamental issues of 'what'.

Nevertheless, the technology is still immature so that the adolescent stage can be roughly characterised by the fact that 'how' and 'what' debates occur about equally, often with regrettably little cross-fertilisation between them. When the product is mature, the technology is fully developed and the market is saturated. All concerned know perfectly well that any relevant product specification can be made so that the crucial questions are entirely concerned with framing a specification that will attract enough users to justify the business. Thus, at this stage, 'what' is dominant. Finally, when the product becomes senile it goes out of use and its social purpose is met by other products.

Fig. 1 summarises this maturing process as applied to steam engines, aircraft engineering and information engines. It suggests that information engines are still in the adolescent stage.

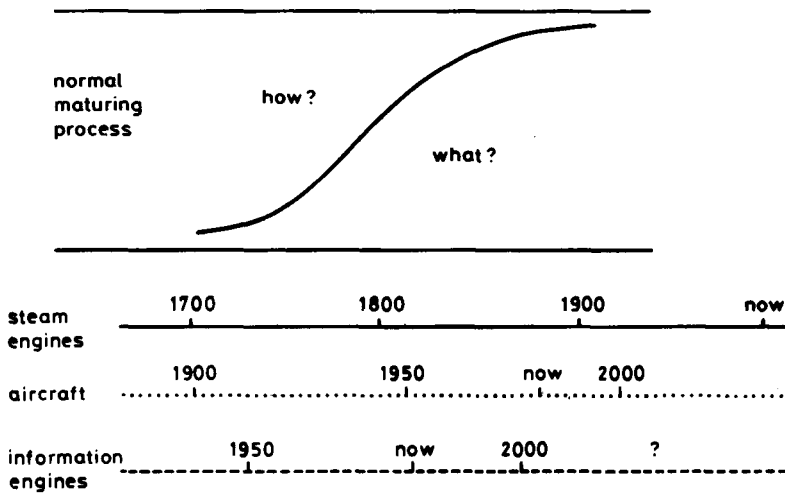


Fig. 1 Technological evolution

The first useful electronic information engine was developed during the Second World War. Since that time, the physicists and electronic engineers have done a splendid job introducing solid-state devices and large-scale-integrated fabrication techniques, which have removed many of the technological constraints that shaped the early information engines. However, the refinement of technology has not yet been complemented by an understanding of the natural properties of information adequate to guide the deployment of our new found technological mastery, so that a first approximation to an understanding of the present situation in information engineering would be to liken it to the situation in the evolution of steam engines in the early 19th century after techniques for casting, forging and machining had provided the 'means' but before theoreticians such as Carnot and Rankine had illuminated the 'ends' for steam-engine design.

4 The present situation in information engineering

Against the foregoing background of general technological evolution it is useful to summarise the present situation:-

- (a) Centralised data processing is common.
- (b) Small decentralised data processing systems are proliferating: these contrasting styles of operation are already being widely discussed.
- (c) Interactive use is costly and fragile.
- (d) Complexity is difficult to control: the complexity of an assembly of hardware or software is essentially measured by the quantity of information required to describe it. Complexity is not the same as multiplicity. Thus, for example, if we look at a semiconductor storage chip through a microscope we see many thousands of components arranged in a highly systematic pattern. Such a storage chip represents high multiplicity but low complexity. On the otherhand, software comprises almost pure information so that it is characterised by pure complexity. Indeed, human limitations in the handling of complexity now control the range of purposes for which information engines can be effectively used.
- (e) Processors originally designed for arithmetic are mainly used for other purposes: we have known this for a long time but have done very little about it.
- (f) LSI is used to reduce cost and improve reliability: these are proper objectives but LSI has not yet been used to improve basic system designs that have changed very little over 30 years.
- (g) Device companies are beginning to enter the systems business by offering naked hardware without much software support.

5 Constructional technology

So much for the present situation; what is going to shape the future? It is easy to summarise the technological situation. The essence of the matter is that planar microfabrication techniques have rendered the constructional units for processor, fast store and 1st-level backing store so similar that all three elements can now be assembled in any mixture that we need. There is no longer any over-riding constructional reason for continuing the traditional practice, pioneered by Von Neumann, of concentrating each type of element—main store, backing store and processor—in a separate box and interconnecting the boxes via bottlenecks. Thus, technological advance permits us to design our systems to meet our up-to-date understanding of the users' requirements but it does not tell us how to do this. Only a deep understanding of the interface between the system and its human users can do this.

6 Essential requirements

It is not so easy to summarise the requirement situation. It is no use trying to analyse the tangled web of computer applications: that way leads only to confusion. Neither is it useful to rake over current systems implementation practice since that

has been done many times already. We must start at the beginning. To achieve an understanding of the nature of information and its role in human affairs, adequate to guide the design of an information system, the study should be regarded as a branch of biology rather than mathematics or electronics.

We must recognise that the human race is a species whose original survival and present dominance is based on the chance discovery by our remote ancestors of a new field for biological competition—the creation and operation of social groups dynamically adaptable to the environment by large scale interchange of information between individuals and groups. We still do this on a grand scale. The hunting and agricultural teams have now become companies and the day-to-day operation of the social co-operation mechanism is called business. Nevertheless, in our business operations we compulsively adopt organisation techniques and associated information handling techniques that served our ancestors for a million years, so that when we introduce a computer system we ignore such inflexible human habits at our peril. However, when computers were first introduced into business, the constraints imposed by primitive electronic technology necessitated that to a great extent the user adapt his practices to the computer rather than vice versa. For example, the centralisation of information processing and its collection into artificial batches represented such a forced adaptation. These practices have continued to the present day and have led to many frustrations that have contributed to the somewhat ambivalent image of the computer and its professional attendants as essential but awkward. We can now recognise that the commercial centre of gravity of information engineering is concerned with database management, where the meaning of the phrase should properly be derived from the human realities. The purpose of a database is to assist communication between the members of a co-operating group of people by maintaining a continuously up-to-date information image of the current state of the group and its relevant environment. This purpose can be served only if all the individuals associated with the database, those who put information in as well as those who access it, feel that the database is of sufficient value to the group and to each individual to justify its cost. In present practice a typical database is not always up to date, it permits its users to ask only stereotyped questions and even to achieve this necessitates a mountain of software that imposes an unacceptable parasitic load on the available computer power and poses a formidable software maintenance problem.

7 Natural properties of human information

It is now possible to see clearly that an effective solution of such problems can be devised only by cultivating an attitude of humility in the design of our information systems. We must first recognise that there is an underlying unity in the tangle of computer applications and that it is derived from their common factor—people.

Information structures appear at first sight to be arbitrary and *ad hoc*; invented on the spur of the moment for each specific purpose. However, such structures are heavily influenced by habits that have been evolved over a long period to create and maintain co-operative social groups. One such habit, the use of tree-type data structures, obviously derived from the wide use of tree-type social organisation structures, is well known to the designers of high level languages but has seldom been reflected in computer design. A consequential and less obvious habit arises

from the fact that for every real situation there are many alternative ways of organising the associated information in relevant tree structures, so that as the situation evolves new tree structures unrelated to those already established tend to be created, grow in importance and then fall out of use. Thus at any time many alternative organisation structures can be said to have meaning to the people whose co-operation gives rise to the information and it is not practicable to represent all such structures by indexes in their database.

A third information handling habit arises from our instinctive preference for the interactive mode of information transfer. Although the human species can legitimately be regarded as an information handling specialist, and we commonly handle large quantities of information, we can best deal with it in small packets with frequent opportunities for checking and clarification. Accordingly, when we access a database we reserve the right to alter our question in response to the information obtained. Similarly, we find it impossible to create a large program free of errors, so that every program needs to be debugged before it can be used, and, indeed, during use. Curiously, some computer professionals tend to take a puritanical view of programming errors. However, the prevalence of errors is simply the human preference for the interactive mode asserting itself and eventually we can expect that a typical information system will be designed from the beginning to be controlled in the interactive mode.

With this view of the social nature of information we can deduce some more natural properties; we can summarise these as follows:

- (a) 'information' bonds society
- (b) information handling habits have been shaped by long use of information to operate social groups e.g. hunting, agriculture, business
- (c) organisation structure of information reflects the structure of the human group that it serves
- (d) organisational structure of information is normally a blend of order and disorder
- (e) people communicate information in small packets with opportunity for checking (interactive mode)
- (f) some information processes naturally occur in batches, e.g. payroll
- (g) most information in the world is not numeric
- (h) a growing proportion of information processing is not arithmetic
- (i) logical operations can be used for all purposes, including arithmetic processing.

8 A technological forecast for information engineering

If we extrapolate from the present situation taking into account the analysis of requirements and available technology that I have outlined, a forecast for the next decade of information engineering is almost obvious. The prime objectives for system design will be data management and keeping complexity under control. The essential technique for ensuring that complexity is manageable is to arrange that the quantity of information handled by the designers or users at one time is within human capability. Thus the design of both hardware and software must be modular. The modules must not be too large and they must be separated by clean and tidy interfaces so that errors in one module cannot sabotage others. Above all, it will be

recognised that universal geniuses do not exist, so that the complexity problem cannot be brought under control by seeking more competent people or appointing a new project manager.

A typical system will be designed to be used interactively by people who are primarily interested in carrying on their own business and have no interest in the technicalities of data processing. Batch processing will be confined to operations which naturally occur in batches e.g. payroll. At the present time attempts to assess the power of a computer system are somewhat confused by empirical and highly artificial units such as the Post Office Work Unit or MIPS. We shall come to recognise that the natural measure of the power of a system is simply the number of people in the organised social group that the system can serve. Systems will be available that will be cost-effective serving only a few tens of direct users and they will be purchasable at a cost that will not require user Board approval. Such small systems will perform functions that can be clearly understood by the user who will be able to interconnect and redeploy his information engines to meet his constantly evolving needs. Intrinsically symmetrical communication techniques will be adopted to make this possible. The proliferation of such small systems will take away some of the load of a typical centralised data processing system so that the data-processing manager will change his role. Instead of taking direct responsibility for all the information processing he will ensure that the small systems distributed over the users' organisation are compatible with one another and can be used to extract corporate information in addition to serving primary roles as departmental information systems.

9 Some ICL research projects

For several years the work of RADDC has been selected to prepare for the situation that I have described. We have put most of our efforts into four projects: variable computer system (VCS), Content-addressable file store (CAFS), distributed array processor (DAP) and interactive man/machine communication by speech.*

9.1 Variable computer system (VCS)

The conceptual origin of this project was the recognition that the natural way that people access information is not by the use of a fixed reference framework, as in the von Neumann machine, but by the use of a reference map that can be created and continuously updated by the user to suit his purpose. The objectives were twofold:

- (a) to demonstrate a working system incorporating low level navigation facilities by means of which a user can create and maintain up to date a secure reference map showing the organisational structure of his information and its mapping on physical storage
- (b) to take advantage of systems incorporating fast microprogram storage by

* Papers dealing in more detail with these and possibly other RADDC projects will appear in subsequent issues of this journal

permitting the target machine to be adaptable on demand to the high level language in which the source program is written.

We now know that the first objective is similar to the 'capability' systems designed at the Universities of Chicago and Cambridge.¹ The provision of a secure map makes impossible many of the programming errors the unmonitored accumulation of which accounts for the severe difficulties that are commonly encountered in the development and maintenance of complex software.

The VCS system has been demonstrable in the Research Department since 1974 and recent work has shown how it could be implemented on standard Company hardware (the MICOS 1 processor in the 2903). We have compared the performance of the VCS/MICOS 1 system obeying COBOL programs with the performance of 1900/MICOS 1 (2903) on the same COBOL programs and find that VCS offers:-

- (a) a reduction in Cobol compiler size in the ratio of 6:1
- (b) a reduction in Cobol object code size of up to 3:1
- (c) an increase of Cobol execution speed in the ratio up to 1:1.8
- (d) a more powerful operating system (in the sense of providing more facilities) which is at the same time more flexible (in the sense of adapting rapidly to different modes of use), with about the same storage requirements
- (e) reduction in main store quotas per job as a result of code sharing and automatic adjustment to working set size.

These measurements, taken as they were under experimental conditions, should be regarded as a spot comparison between an existing COBOL implementation and a first version of an implementation via VCS. Doubtless both could be improved.

The complete technique of creating, maintaining and using an information map can be summarised as 'access by navigation'. This navigation technique is of necessity used in all existing computer systems but in most of them the map is specified completely only in the source version of the program and is irreversibly confused at the time when it is most needed, that is after compilation. The VCS system preserves the information map explicitly at run time so that accidental deviations from authorised paths on the map can be immediately monitored and controlled. The intrinsic security that these mechanisms permit operates within as well as between programs and serves all the software, including the system software, at trivial cost.

Evidently the use of the navigation technique requires that the work involved in creating and maintaining the map be small compared with the work representing the primary purpose of the information system. This was the case for the tasks for which computers were first used. However, of late it has been increasingly recognised that Data Management can be expected to become the most significant use of information systems. A database, by its very definition, represents in information terms the activities of people whose interactions cannot be totally predictable. It follows that the maintenance of the information map of a database involves a very great deal of work to ensure its continued integrity and accuracy as a true representation of the network of agreements, promises and achievements that bind a co-operating group of people. This situation has become widely recognised by those who have been trying to devise standardised navigation techniques for data management. Indeed, this recognition no doubt accounts for the controversial nature of standardisation proposals such as Codasyl. This problem is a fundamental conse-

quence of an intrinsically unpredictable component of the natural behaviour of human information so that it is unlikely ever to be overcome within the limitations imposed even by an efficient and secure navigation technique such as used in the VCS system. It is therefore also necessary to provide a complementary technique to retrieve information by search in those circumstances when the work involved in maintaining the map up-to-date is either excessive, or, in some cases, intrinsically impossible. There is, therefore, a concomitant requirement for a cost-effective technique for accessing information by searching for it. The CAFS project was undertaken to explore ways of providing such a facility and using it in combination with the navigation technique.

9.2 Content-addressable file store (CAFS)

Over the past few years we have designed and built such a searching device based on the use of disc files and have conducted extensive experiments on methods of using this to meet difficult requirements such as telephone directory enquiries, biblio-

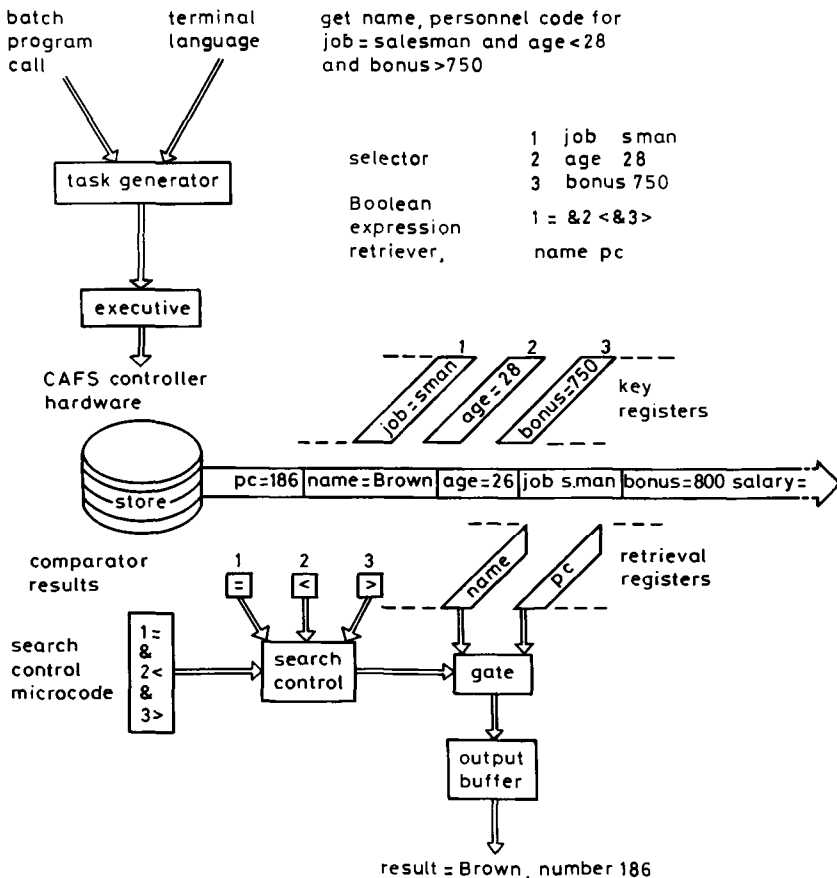


Fig. 2 Content addressable file store

graphic information retrieval and management information systems. The primitive operations carried out by the searching engine are illustrated in Fig. 2. A selection function is formed from a description of the required data unit that has been issued by a terminal message or by a program. The encoded selector is passed to a backing-store controller equipped with scanning hardware which comprises key-matching channels operating simultaneously on a stream of data and a special processor that evaluates Boolean or threshold expressions into which the outputs from the comparator enter as arguments. A wide range of common types of selection function can be represented in a very direct manner in the 2 level evaluation hardware. It is possible for more than one independent search task to be active on the same data stream, the major constraint being the number of key channels available. The latter plug into a standard highway system and may be thought of as a variable resource analogous to mainframe storage. Each channel is capable of detecting relevant data fields by matching against embedded identifiers, and comparator masking is available to permit part fields to be isolated.

Retrieval of data from 'hit' records can be achieved selectively by collecting the contents of designated fields. It is therefore possible to compose virtual 'reply records' comprising only those data required by the calling process, arranged in a specified sequence. Alternatively, a count of 'hits' may be all that is required, in which case no data as such are recovered. This editing of results is particularly valuable in minimising central resource loading in interactive situations and is generally beneficial in view of the size to which multipurpose records can grow.

The hardware control facilities are completed by normal physical device controls, including write channel administration. Overall organisation is effected by a mini-processor that is also available to provide a further level of data sieving and composition. The overall balance of search hardware is therefore seen to comprise a filtering mechanism in which the full backing store transfer rate is handled only by very simple, repetitive hardware, with progressively more complex operations being performed on successive abstracts of diminishing volume, culminating in procedures executed in the mainframe. Thus the sum of the products of data rate and complexity of operation is minimised, and, in particular, the mainframe mill and backing store channel load can be reduced by several orders of magnitude compared with conventional serial file processing.

The storage medium can be serial or block accessed, such as tape or disc. The most generally useful device is the magnetic disc. On a typical high-capacity disc handler many heads are available that require only the addition of some fairly inexpensive electronics to provide a greatly increased readout rate. Such a high rate would flood all but the most powerful central processor, but the progressive abstraction scheme of the special scanning equipment described above renders high speed searching entirely feasible.

Our studies of the use of autonomous file searching devices have shown that it is quite practicable to implement a relational model of a database. Indeed, we have found it possible to refine the relational model beyond the published work of Codd and his associates. As I have explained, this project originated in a conscious attempt to identify the facilities required for a database management system and to provide them by taking advantage of up-to-date technology. We have now reached a stage in which we have studied a variety of specific manifestations of the generalised data base problem and we have not yet discovered any reasons for changing our view of the intrinsic nature of the database management task.

9.3 *Distributed array processor (DAP)*

This project was originated about 5 years ago in direct response to the recognised requirements of weather forecasting and meteorological research. Since that time the scope of the work has been broadened to include a very much larger range of problems, for example plasma physics and associative information retrieval.

The great majority of present computer systems can be regarded legitimately as direct descendants of the Von Neumann machine in the fundamental sense that they are characterised by four features:

- (a) the processor is primarily designed for arithmetic operations, with logical operations regarded as a byproduct
- (b) the store and the processor are separate
- (c) the store-processor combination can obey only one instruction at a time, each requiring not more than two operands
- (d) the essential objective in programming such a machine is to represent the overall task by a serial string of such instructions.

These primary features have been somewhat blurred in some powerful machines by tactical measures such as pipe-lined operations on ordered strings of operands, but the fundamental principle that strings of individual instructions are obeyed sequentially is still valid. Hence the connection between store and the processor inevitably imposes a well defined upper limit to the rate at which the whole assembly can operate. In short, it is a bottleneck.

With the introduction of semiconductor storage and large-scale integrated circuits, the original reasons for separating processor from storage are no longer valid. Furthermore, most of the information in the world is not numeric and consequently a growing proportion of computer operations are not arithmetic, so that we should now regard arithmetic processing as a specialised use of more general and fundamental logical operations. Accordingly, to deploy up-to-date technology to meet a broad spectrum of users' requirements, it is now appropriate to revise all four features of established system design practice. In the DAP all these changes have been made.

The conventional semiconductor store inevitably is made in many elements, each typically storing a few thousand bits. In the DAP each element has its own very simple processor primarily designed to carry out logical operations on one bit operands. It writes its results into its own storage element and can use, as input, information from its own store, its immediate neighbours or elsewhere via row or column highways in a matrix of storage elements. Thus the DAP offers the following fundamental advantages over current methods:

- (a) the simple processing elements are easy to design and build, and are flexible in use
- (b) the physical distance between each processor and its store can be very short
- (c) there can be many such store-element/processor-element connections operating simultaneously
- (d) real problems are commonly parallel by nature: the DAP provides a parallel processing capability that can match the structure of the solution to that of the problem. The DAP Fortran language gives a concise

and straightforward way of expressing parallel operations and has already been used to program several applications on the pilot DAP.

- (e) since the processing elements are primarily designed to carry out logical operations, a valuable speed-up factor compared with present practice is achieved on all operations, data manipulation as well as arithmetic operations. A typical present computer system tends to spend more of its time on data manipulation than arithmetic, so that the DAP offers a substantial performance advantage on a wide range of applications.

All the processing elements obey a common program but each element can be instructed or instruct itself to ignore any command in order to provide sufficient flexibility to enable the apparently rigid matrix to be adapted to the needs of different problems.

The complete assembly can be regarded as a store that has all the properties of a traditional store but with the extra facilities that have been described. It can therefore store its own instructions in the normal way. Moreover, the DAP store can be incorporated as part of the store of an existing host computer of conventional design that is responsible for putting the problem in the DAP part of its own store and also obtaining the answers. In this way it is possible to take advantage of the power of the DAP to tackle difficult processor intensive parts of real problems without requiring complementary development of a new operating system.

The proposal was conceived in 1972 and a 32 x 32 pilot machine has been working in RADC since 1976. This has given firm information on basic performance and has made it clear that the DAP can be applied to a wide range of information processing tasks; and indeed that intrinsic serialism in real problems is quite rare. Table 1 shows estimates of the performance of a 64 x 64 machine, derived from detailed studies, for a number of complete calculations, as compared with existing implementations on particular machines.

Table 1 Performance of 64 x 64 machine

Arithmetic-dominated programs

Structural analysis (finite-element method)	6 x IBM 360/195
Meteorology (complete suite of operational programs)	13 x IBM 360/195
Many-body problem (galactic simulation)	10 x CDC 7600
Magneto-hydrodynamics (3-dimensions)	14 x IBM 360/91

Decision-dominated programs

A table look-up problem	3 x CRAY-1
A pattern-matching problem (binary strings)	300 x IBM 360/195
Operational research ("the assignment problem")	1200 x IBM 370/145

More information on the DAP is given in the Bibliography. With a longer-term view, the DAP can be regarded as a new system component, a store with built-in processing capability; this is likely to have far-reaching effects on the evolution of systems engineering practice, as for example in the efficient implementation of distributed systems.

A 64 x 64 system has been ordered by the Computer Board for installation at Queen Mary College, University of London, where it will be attached to a large 2980 host, and the Company is now actively marketing the DAP as an enhancement to its standard system products.

9.4 *Man/machine interaction by speech*

A clearly recognisable human habit in the communication of information is the preference for the 'conversational' mode. It can be regarded as a behavioural adaptation to the imperfections of human communications, since each individual message can be supplemented on request by repetition or clarification.

To carry on a conversation each participant must be able to reply quickly, before the last speaker has forgotten what he said and why. The CAFS system permits such rapid interaction using a keyboard and video display for man/machine communication. Now that we have a machine that can respond fast enough to be a credible conversational partner, the ultimate objective is man/machine communication by speech to permit conversational working in the full sense of the word. This is a most difficult problem. The process of human speech communication by natural language is not fully understood, and certainly could not be reproduced by a machine. Our objective is to develop techniques that will enable practical speech communication with an information system to be effected retaining the major advantages of using speech: its ease of use and its efficiency as a means of information transfer.

It is perhaps useful to consider the requirements for a speech input/output system. It should be based on the use of an ordinary telephone to avoid expensive terminal equipment and to allow potential access to a very large number of users. It should incorporate standard digital technology, which, on account of its intrinsically high speed compared with the information rate of speech, can be multiplexed to achieve low cost per channel. The use of digital techniques brings the usual advantages of reliability, repeatability and maintainability to what has traditionally been the province of analogue techniques.

The recognition device should be adaptive so that it compensates for peculiarities of the speaker and the individual telephone. In addition to the vocal adaptation the overall system should be designed to simplify the recognition problem by taking advantage of context. At each stage of successful communication in a conversation, the possible repertoire for the next communication is often known to be restricted and there is every reason to take advantage of this fact. By such means it is possible to match a simple machine to a human user capable of great subtlety without excessively annoying the user. Indeed, it is possible to a limited extent to make the machine detect whether a user is experienced or casual and untrained, and structure the interaction accordingly.

Speech output is an easier task for a machine than speech input and is likely to be of commercial significance sooner. We now have ready for exploitation a speech synthesiser, a powerful technique for speech output that can be implemented either in multichannel or single-channel form. This development has great potential and will enable computer based information services (interactive and noninteractive) to give direct spoken information to the public. For example, in a directory enquiry system, about 10 s of operation connect time is spent in relaying the telephone number to the enquirer. The use of a speech output device for doing this job would save an estimated £2,000,000 a year in the UK.

Our research activities in the speech interaction field have inevitably caused us to be more aware of the intrinsic nature of information as a byproduct of human life. This has been most valuable and will help us to develop techniques for handling the more complex input/output that will be a system requirement of the future.

10 Summary and conclusions

Prognosis for information engineering

- (a) A typical information engine will be conceived as a subsystem to a natural human information system. This needs a little explanation. What do we mean by system and subsystem? In conventional usage we think of a computer as a system and its peripherals as subsystems. We used to think we could make a computer any way we liked but a peripheral device must be constrained to plug in to the computer. We can now recognise that the computer itself is a subsystem in this sense, whose existence can be justified only if it is consciously designed to serve people whose behaviour is unnegotiable. All this will necessitate a new attitude of constructive humility to be adopted by information engineers.
- (b) We must exploit the order that users instinctively impose on their information and respect the disorder that arises from the fact that human affairs are not totally predictable. We are already well practised in exploiting order since the design and use of high level languages is essentially directed to this end. However, we can expect to gain much advantage by providing the means for exploiting order at the lowest practicable level in our information systems so that they can offer advantages for much of the system software as well as for the ultimate user. In present practice, disorder is not respected and is too often regarded incorrectly as a failure of overall system design.
- (c) Technological advance permits such a system to be designed but does not guide how to do it. Some relevant techniques have been demonstrated in ICL research.
- (d) Profound changes in information system practice are inevitable as a consequence of *a*, *b* and *c*.
- (e) However, the timing of such changes is difficult to predict since large scale events are controlled by a commercial stick/slip mechanism. This arises from the fact that all large scale decisions are quite properly made to maximise return on investment. The total situation evolves by the accumulation of understanding of objectives together with ripening technology. When such reasons for changes are less than decisive no changes occur at all, since the right business decision is to obtain some more return from existing investment. When eventually the reasons become decisive, a bandwagon effect occurs, so that the changes occur more quickly than might be expected. In my judgement the 'slip' is likely to occur in the next decade.
- (f) Genuinely modular system design will be increasingly practised and will lead to de facto standard functional specifications for modules.
- (g) A typical module will comprise a combination of hardware possibly including active storage modules derived from the distributed array processor and software whose overall functional specification will be clearly understood by the user in ordinary human terms.
- (h) The user will be able to control the deployment of such modules to match his evolving requirements.

Bibliography

Variable computer system

- 1 DENNIS, J.B. and VAN HORN, E.C.: 'Programming semantics for multiprogrammed computations' CACM, 1966, 9
- 2 ENGLAND, D.: 'Capability concept mechanism and structure in system 250' IRIA International Workshop, Protection in operating systems, 1974, pp. 63-82
- 3 EVANS, O.V.D. and MAY, J.: 'The VCS system - an overview of its operational aspects' VCS.20 RADC, ICL, 1975
- 4 FABRY, R.S.: 'Preliminary description of a supervisor for a machine oriented round capabilities' ICR Quarterly Report, 1968, 18, University of Chicago, Section 1B
- 5 FABRY, R.S.: 'List structured addressing' Ph D thesis, University of Chicago, 1971
- 6 ILIFFE, J.K. and JODEIT, J.G.: 'A dynamic storage allocation scheme' *Comput.*, 1962, 5, pp. 200-209
- 7 ILIFFE, J.K.: 'Basic machine principles' (MacDonald/American Elsevier, 1968)
- 8 ILIFFE, J.K.: 'Basic Machine language' TR.1021 ACTP final report. RADC, 1969
- 9 NEEDHAM, R.M.: 'Protection systems and protection implementations' *Proc. AFIPS*, 1972, 41
- 10 NEEDHAM, R.M. and WALKER, R.D.M.: 'The Cambridge CAP computer and its protection system' SOSP6, 1977
- 11 SALTZER, J.H. and SCHROEDER, M.D.: 'The protection of information in computer systems', *Proc. IEEE*, 1975, 63
- 12 Variable computer system' ACTP Final Report. TR. 1157 RADC, 1974

Content-addressable file store

- 1 CODD, E.F.: 'A relational model of data for largeshared data banks', *Comm ACM* 13, 1970
- 2 COULOURIS, G.F., EVANS, J.M. and MITCHELL, R.W.: 'Towards content addressing in data bases' *Computer J.*, 1972, 15
- 3 LIN, C.S., SMITH, D.C.P. and SMITH, J.M.: 'The design of a rotating associative memory for relational database applications', *Trans. Database Syst.*, 1976, 1, pp. 53-65
- 4 MITCHELL, R.W.: 'Content addressable file store' Online Database Technology Conference, 1976
- 5 OZKARAHAN, E.A., SCHUSTER, S.A. and SMITH, K.C.: 'RAP - an associative processor for data base management' AFIPS 1975 National Computer Conference, 44, pp. 379-387
- 6 SU, S.Y.W. and LIPOVSKI, G.J.: 'CASSM: a cellular system for very large data bases' Proceedings of the ACM International Conference on very large data bases, 1975, Framingham, Mass., pp. 456-472

Distributed array processor

- 1 FLANDERS, P.M., HUNT, D.J., REDDAWAY, S.F. and PARKINSON, D.: 'Efficient high speed computing with the Distributed Array Processor' in 'High speed computer and algorithm organisation' (Academic Press, 1977), pp. 113-128
- 2 PARKINSON, D.: 'An introduction to array processors' Systems International, 1977

Man/machine interaction by speech

- 1 ADDIS, T.R.: 'Human behaviour in an interactive environment using a simple spoken word recogniser' *Int. J. Man-Machine Studies*, 1972, 4, pp. 255-284
- 2 ADDIS, T.R.: 'Human factors in automatic speech recognition' RADC, ICL. Technical Note TN. 78/1
- 3 UNDERWOOD, M.J., ADDIS, T.R. and BOSTON, D.W.: 'The evaluation of certain parameters for the automatic recognition of spoken words, machine perception of patterns and pictures' Institute of Physics Conference Series, 1972, 13, pp. 117-125
- 4 UNDERWOOD, M.J. and MARTIN, M.J.: 'A multi-channel format synthesiser for computer speech response' Proceedings of the Institute of Acoustics, Autumn Conference, 1976, 2-19-1
- 5 UNDERWOOD, M.J. 'Machines that understand speech' *Radio & Electron. Eng.*, 1977, 47, pp. 368-376

Standards for open-network operation

Jack Houldsworth

Manager, ICL Letchworth Development Centre

Abstract

The introduction of data networks demands the consideration of open operation between subsystems of mixed origin and highlights the importance of overall interchange standards. Work on international standards for interactive, remote job entry and file transfer has begun, but for the present we will have to live with the existing commercial protocols and map these on to mainframe and terminal systems. This paper describes the architectural approach to the design of high-level protocols and the short- and long-term prospects for standardisation. The overriding conclusion is that the economic value of standards in the open-network environment is indisputable.

1 Introduction

Until quite recently, the traditional computing scene was dominated by installations that were planned and implemented as self-contained, 'closed' systems. Some resource-sharing systems with elements from several different commercial origins have been made to operate successfully but the agreement of the rules for interconnection have required delicate negotiation, and, more often than not, the complete adoption of the rules of the dominant supplier in the mix. Even these systems, for all intents and purposes, operated in a closed-system environment.

A significant change can now be detected as some users begin to explore the possibilities of 'open working'. This is defined as the ability of the user or the program of any computer to communicate with the user or the program of any other. The awareness of the possibilities of open working is a direct consequence of the planning for data networks which has been carried out in many countries over the last few years. Data networks will follow the same pattern of evolution as voice networks and they will have the same degree of 'openness' as the telephone and the Telex networks, with complete freedom of interconnection.

The International Telegraph & Telephone Consultative Committee (CCITT) has defined a standard network-access protocol (X25), which was originally aimed at packet-switched networks, but it will undoubtedly be offered by all future packet- or circuit-switched data networks. It is widely believed that this new protocol will immediately solve all interconnection problems, once the basic hurdle of introducing the interface has been overcome. This is not so! The interface and the data networks behind it will be transparent to the interchange between users. The need for mainframes to be aware of how alien subsystems are driven is un-

This paper is an updated version of a paper published in *Computer Communications* in February 1978. We are grateful to the Editor and the publishers for permission to use this earlier material

changed, whereas the additional requirement for mainframes and terminals to be able to control the path through the network is introduced.

It is a fairly straightforward task to adapt mainframes to drive X25, but it is not practical to adapt all the existing terminals that users may want to connect to networks. Hence some way of attaching terminals has to be found and the popular idea is to define 'virtual-terminal' protocols. These will be a synthesis of the common features of each popular class of terminal. Fringe conversion units will take care of the differences between real terminals and the virtual-terminal standard. Virtual-terminal standards will only solve the basic communication problem and, even in the simple case of a 'scroll-mode' teletype, the communicating mainframe will have to be aware of the specific code set, the number of characters per line and any special tabulation facilities at the physical terminal before sensible information can be transferred and printed.

This begins to indicate the problem. The scroll-mode terminal is only the beginning. The problem becomes more complex as keyboard/screen and remote-job-entry (RJE) terminals are considered. In these cases a sophisticated dialogue with the mainframe operating system is often involved. The next major step towards open working will be the free interchange of files. In addition to common file structures, this will require standards for the dialogue between the operating systems of the communicating mainframes.

The International Organisation for Standardisation (ISO) and the CCITT have generated standards in many of the areas involved but now a great leap forward is needed to bring these existing standards together into a properly structured architecture and to add the elements that are necessary to transport information freely in the open networks of the future.

2 What is an open network?

Fig. 1 gives an impression of an open network of the future and the broken line shows typical intercommunication paths. In a truly open concept, each terminal, small processor and host processor will be able to intercommunicate freely and may agree to delegate, accept or share tasks. Even the systems that were previously regarded by the CCITT as message-transfer services, such as facsimile and the proposed enhanced Telex service (Teletex), are coming under the open network umbrella. The latter is bound to grow and merge with the general word processing environment. Ultimately, all data-communication services will be enveloped.

3 The architectural approach

It is essential that a strict architectural approach to the design of the overall communication process is adopted. Using this approach it is possible to hide the particular facilities of each network from the higher levels, thus allowing the terminals and processors to use the communications facility as a subnetwork that is totally transparent to the actual data-communication process. Common high-level application-oriented protocols, which are network independent, can only be designed when this approach is used.

All the functions involved in the overall communication process can be broken down using the 'onion-skin' architecture technique. This was developed during the original work by ICL on data-communication protocols which led ultimately to the

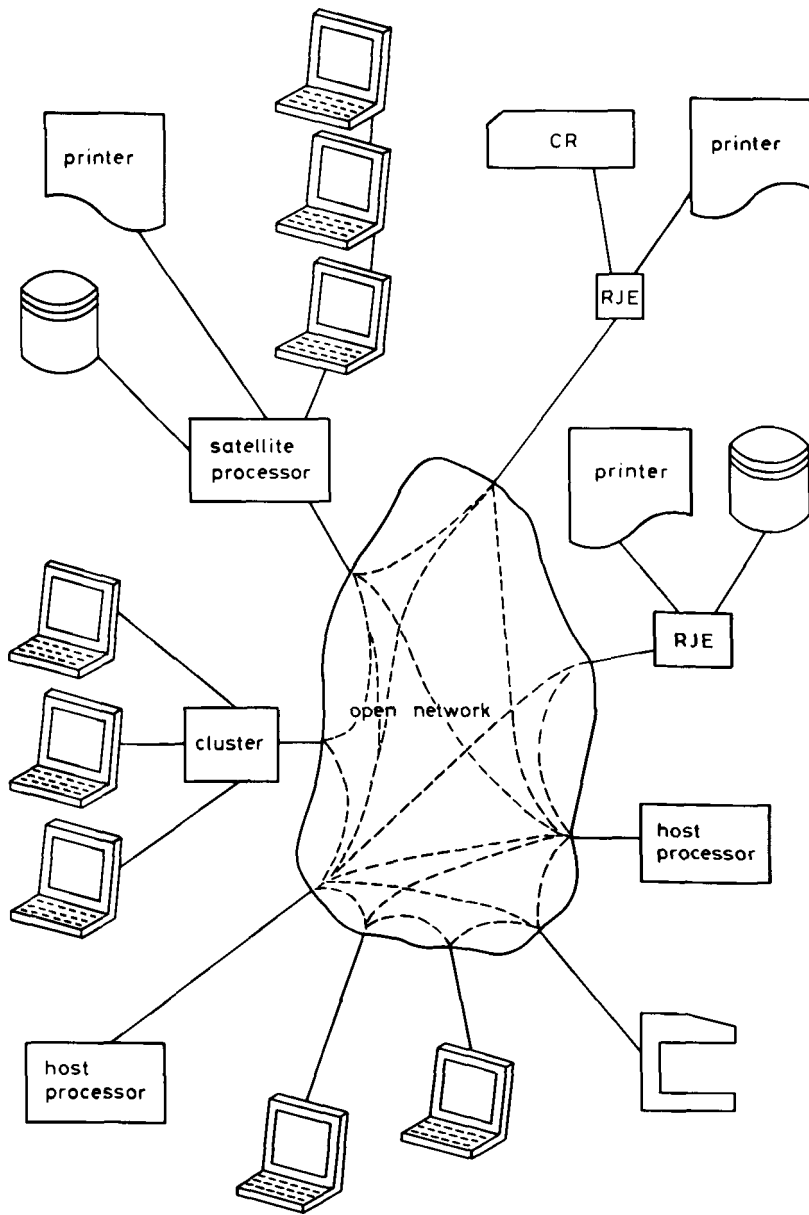


Fig. 1 Typical open network

current high-level data-link control (HDLC) standard.

ICL recognised that each function in a communication process can be physically or logically separated and that any function that is performed on entry into a system is exactly reversed on exit. Hence the system is mirrored about the centre and the hierarchical layers can be drawn as concentric circles; hence the term

'onion skin'. When this principle was synthesised by Mr. D. Ackerman of ICL in 1968 it was called 'the principle of complementary symmetry'. The current advanced status of this work will be exposed in a later issue of this journal.

Fig. 2 represents the overall communication process in computer networks. Typically, the user at one computer will send commands and data down through these functional levels, across the telecommunication network, and up through corresponding functional levels in the remote processor. The response is in the reverse sequence.

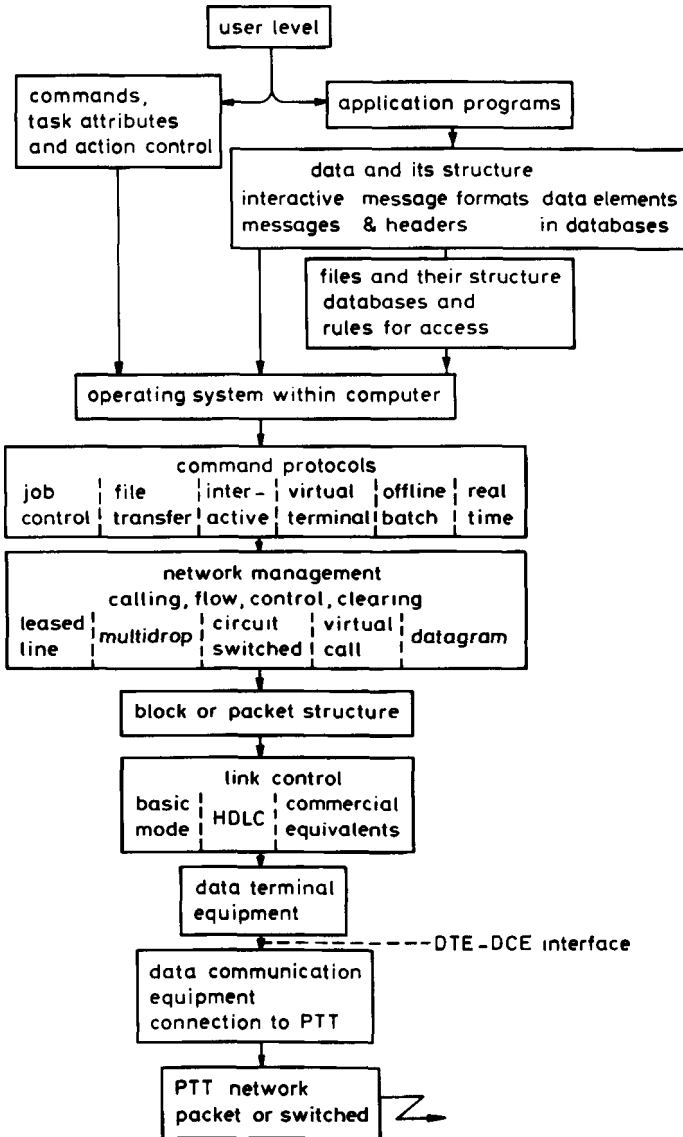


Fig. 2 Architecture of communication process

4 Definition of terms

User Level: the user himself or a program in a computer that initiates or controls a transfer of data either manually or automatically.

Commands: the commands that determine the type of communication, the attributes or characteristics of a particular transfer and the control of actions.

Data and its structure: the coding of information, the formats of messages and the creation and display of messages, especially in interactive operation where a terminal is involved.

Files and their structure: database structures, the format of stored data, the means to access files, sorting sequences and file labelling.

Operating system: the basic software in a processor that controls all processes and operations. Virtually all processes are routed through this. Access methods must blend with all the proprietary operating systems.

Command protocols: currently three major types are recognised. These are protocols to handle files of data (including the activation of input/output devices), job-control command protocols (including the loading of new programs into a remote computer), and message protocols, especially for interactive operation. Message protocols will include the virtual-terminal protocols mentioned above.

Network management: the method of establishing a transmission link between sender and recipient. This may involve switching, including the addressing of the sender and the recipient, and the establishment of the identity of the calling and called parties. It also includes flow control, recovery from line errors and faults, alternative routing, means for internetwork operation, priority, security and the method of terminating a call.

Block or packet structure: the method of distinguishing line-control signals from data by adding some form of 'envelope' with a header (to include addresses and other link-control data).

Link control: once a transmission link is established, its control, including error control and retransmission, is referred to as link control. The basic mode and the HDLC procedure standards that have been produced by ISO are typical of this level.

Coupling equipment: the means within the data terminal equipment (DTE), which may comprise a simple terminal or a whole computer, to connect to the line. This may include the connection to concentrators or multiplexers.

Connection to the PTT network: the PTT equipment is often referred to as the data-circuit terminating equipment (DCE). In existing systems this will be a modem but new data networks may have a unit known as a network terminating unit (NTU). Transmission, timing and synchronisation are provided at this level. It will also include the circuit into the local concentration point or the switching centre of the network.

PTT network: includes all the switched or leased network facilities provided by the PTT. In packet-switched systems it includes all the exchange equipment which controls the calls and the packet switching exchanges or nodes. It also includes user-support services, such as testing and fault-finding facilities and the monitoring of the performance and the quality of the service.

It may be noted that in some cases certain of the above levels may be bypassed (e.g. an interactive message may not involve a data file) or the route may be predetermined (e.g. point-to-point communication over a leased circuit).

5 The value of an architectural approach

There are three main views of a computer network: the administrator looks at its value to his users, the computer man sees only the computers and the telecommunicator looks outwards from the network. The architectural concept clearly relates each function and permits these three viewpoints and perspectives to be combined.

The architectural diagram defines the chain of functions that make up overall process. If any link in the functional chain is weak—or, worse still, omitted—the efficiency of the whole system is impaired.

Fig. 2 reveals clearly where interfaces should be established (the term is used with its proper meaning of an imaginary surface at which the form, function and procedures for signals that cross the interface are specified). They should lie between an adjacent pair of the functional units.

The approach also shows 'regions' where standards may be prepared effectively—to fall within one or a few adjacent functional units and fit neatly between pairs of interfaces.

6 Implementation

It is emphasised that the levels of Fig. 2 are functional levels. Some of them correspond on a one-to-one basis with items of equipment, e.g. 'connection to PTT network'—to a modem and a circuit to the exchange; others correspond to identifiable items of software, e.g. the operating system.

There are others, however, that have to be grouped together for implementation and Fig. 4 shows a typical example of the arrangements for one user at a simple terminal to communicate either with another similar user, via the operating system of a host processor, or directly with a disc-based service in a remote host computer. In this, the functions between 'network management' and the 'DTE coupling', inclusive, are termed the 'transport service' and those from 'connection to PTT network' to the telecommunications facility are termed the 'telecommunication function'.

In most data networks the preferred interface between the transport service and the PTT network will be CCITT, X25, which is described below. However, simple unbuffered character-mode terminals, such as teletypes, will not be able to handle messages in the form of synchronous packets and the transport-service function must include a packet assembly and disassembly (PAD) level which can be provided either by the user, the terminal supplier or the network implementer. The interface between the PAD function and the network can then be at the X25 packet level. The PAD function ensures that characters can be transferred between the character-mode terminal and the packet network but the actual control of the mechanism and the display formats are a bigger problem.

Network implementers are busily defining the basic functions of the ideal character-mode terminal and this notional terminal will be known as the 'virtual terminal'. Users will expect to encounter this virtual terminal, and any physical terminals that do not conform will be handled by mapping at the command protocol level. Certain basic functions such as speed control and specific control characters will have to be incorporated at the command protocol level in the PAD function at the terminal end of the link. These functions are shown as VTP in Fig. 4.

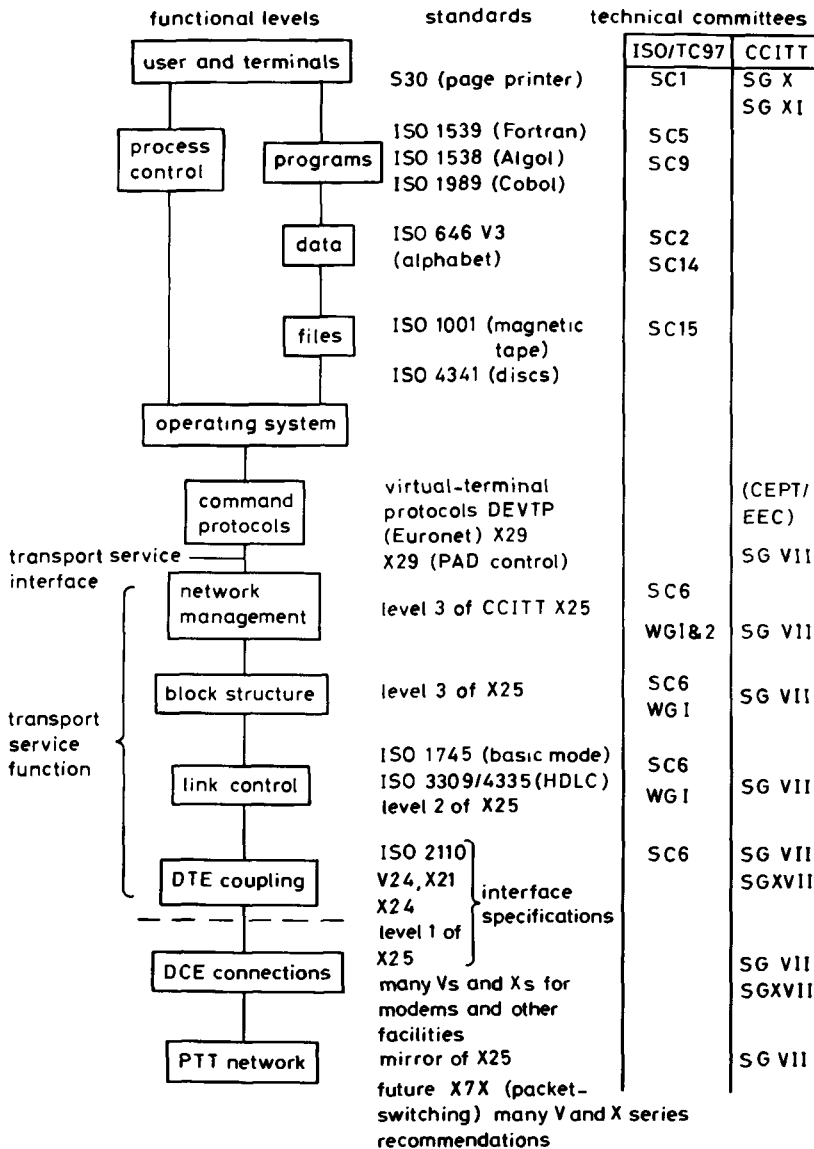


Fig. 3 Standards organisations involved

The other function that is shown at the command protocol level in Fig. 4 is the file-transfer protocol requirement (FTP). If the FTP characteristics are different in the two hosts more mapping will be required at the command level to interpret the commands that control the interchange and it may be necessary to carry out even more mapping conversions at other levels such as the file structure level.

7 Progress towards standardisation

The following Sections outline the progress that is being made by ISO, CCITT and the network implementers, and highlight the important network standards. It is important to distinguish between a genuine attempt to create standards that fit the architectural concept and those that are produced to handle the existing population of terminals and processors.

8 Data-link control procedures developed by ISO

These are in two generations, both of which were originally aimed at star-connected, multipoint links, with centralised mainframe control.

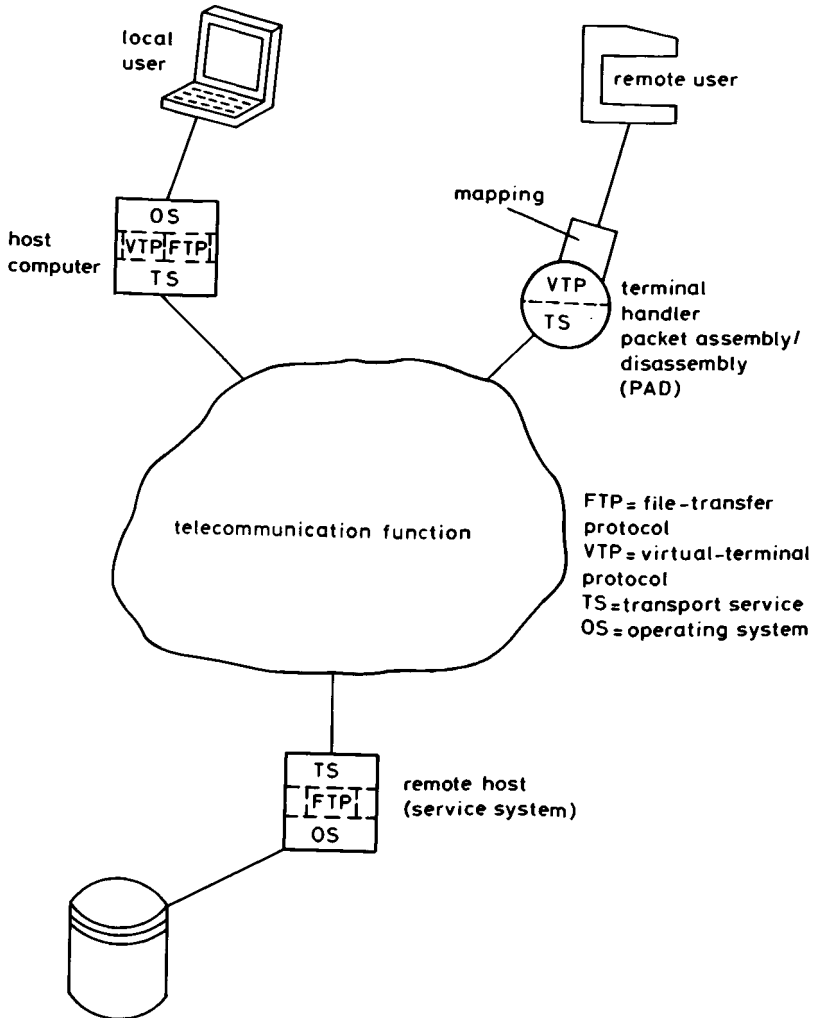


Fig. 4 User communications.

8.1 Basic Mode

The first generation, known as 'basic mode', was produced between 1962 and 1973. This is a character-oriented protocol which uses transmission-control characters to identify significant events in the flow of data and control information.

Basic mode defines only the data-transmission mechanism and is intended to be 'more or less' transparent to the data that are being transmitted.

Most manufacturers have developed basic-mode compatible equipment. They all used the same generic standard but the interpretations have been different and most implementations embody terminal and device-control functions. It is likely that the best features of the most popular interpretations will be lifted by the network implementers and regarded as virtual protocols for interactive and RJE operation. This is discussed later under virtual-terminal protocols.

It should not be forgotten that since the basic-mode protocol defines only the link-control aspects of the system it can be fitted very neatly into the architectural pattern described above. Indeed the existing ICL architecture was developed in association with the basic-mode standard and the higher levels will be transported to other link-level protocols, such as HDLC.

8.2 High-level data-link control

The second generation data-link control protocol, known as high-level data-link control, emerged as a by product of the architectural studies described above.

HDLC has only one transmission delimiting sequence and transmission is always synchronous, which means that the link-control hardware can be produced in a standard form for all connections.

HDLC was originally designed to cover mainframe-to-mainframe and mainframe-to-terminal applications, and was specifically aimed at 2-way simultaneous transmission over point-to-point or multipoint leased circuits. One station controls the scheduling of the link and all other stations are secondaries. The standard is in two parts:

- (a) A 'Frame-Structure' standard (IS3309) rigidly defines a transparent envelope that is used for all data, control and response messages. Each envelope contains space for an 8 bit address and 8 bits of link-control information.
- (b) An 'Elements of Procedure' standard (IS 4335) defines the commands, the responses and the frame-numbering system which are all included in the 8 bit control field of the standard frame.

The 'Elements of Procedure' standard contains several options and the user must select the appropriate parts for his particular application. To limit the variety of implementations, and to improve the chances of compatibility between systems, ISO has generated two 'class of procedure' standards, for point-to-point and multipoint operation, respectively. These are still passing through the ISO voting process but the one that is of direct interest in the network context is the point-to-point version, which has been adopted by the CCITT for the link level of the X25 network-access protocol.

9 Synchronous access to packet-switching networks

The CCITT has produced a Recommendation (X25) for synchronous access to packet-switching exchanges. (Note that the CCITT produces recommendations not standards). This covers the DTE to DCE control levels in Fig. 2 and is divided into three logical levels:

- (a) Level 1 is an electrical and procedural DTE-to-DCE interface.
- (b) Level 2 is a rigidly defined point-to-point HDLC class of procedure for transmission and error control between the DTE and DCE. Currently two versions of level 2 exist. Some changes took place in the ISO philosophy after the CCITT prepared its original standard and the CCITT has now introduced a second version, which is compatible with the latest ISO point-to-point standard. It is expected that the original version will eventually fall into disuse.
- (c) Level 3 defines how packets are transmitted inside HDLC envelopes using heading formats to define the packet type, e.g. network-control packets (call set up and clear down, call progress information, facility control etc.) and user-to-user data packets. The heading includes the subscriber addressing information.

A 'virtual-call' approach is adopted, which means that an establishment phase precedes the data packets and a virtual path is set up for the subsequent data-packet interchange. The heading of each packet contains provision for logical channel identifiers for multiplexing a number of calls on the same DTE-to-DCE path. The need to include the addressing information in each data packet is avoided by referencing the route to the logical channel number which is used in the call-establishment packet.

If the system that is being implemented is the equivalent of a leased-circuit star network a logical channel can be permanently assigned to each possible connection to avoid setting up and clearing down the virtual circuit for each interchange. This is known as 'permanent-virtual-circuit' operation.

A 'datagram' enhancement to X25 is being studied. If this is agreed, it will allow data and routing information to be sent in the same packet to avoid the overhead of call establishment and clearance for short data messages.

Most PTTs and many other network implementers have already declared their intent to offer X25 on their future national networks. The UK Post Office and Euronet are amongst these. The Datapac network in Canada is already offering X25, but at the moment this implementation uses the old version of level 2. The French network Transpac also has X25 as the standard interface for packet-mode terminals. There is no doubt that X25 is the access protocol of the future and we can now look forward to enhancements within the framework of the standard, such as the datagram facility. These new facilities will have to be hidden from the higher levels of the system, within the transport function, to ensure that the high-level protocols, when produced, can retain their truly network independent characteristics. This will avoid the need for the network idiosyncrasies to be known by the main operating system of the host.

The CCITT and the ISO have identified a need for a simplified access protocol for synchronous HDLC systems. The work was commenced under the heading of

'Frame-mode DTE', which has now been changed to 'New common DTE/DCE interface'. The studies are at the early definition stage.

10 Network transport protocols

The node-to-node transport protocol within a data network is invisible to the user except for the end-to-end delays which are introduced. Standards in this area are always regarded as the business of the carrier, who may use time-division multiplexing, circuit switching or packet transmission to achieve what he believes to be the most effective system. Sometimes the choice is political and is only loosely related to economics. 'Gateways' will exist between networks to take care of any differences between transport protocols, and these should also be invisible to the users. However, it should be noted that a fully comprehensive gateway must also take care of the mismatches between codes and higher-level protocols. This is mentioned later under virtual-terminal protocols.

It is generally thought that the CCITT X25 recommendation covers the network-transport protocol area, but this is not the case and the CCITT is producing a specific recommendation for this function.

11 Virtual-terminal protocols

Virtual-terminal protocols are being developed by the CCITT, Euronet and others with loose co-operation and much overlap. These recognise that certain classes of terminals already exist and attempt to map the popular classes and their current protocols on the packet-switched networks.

Virtual-terminal protocols aim to create a situation where all mainframes know the characteristics of a standard notional terminal and the differences between the virtual terminal and the physical terminal are sorted out by conversion equipment to produce a sensible presentation. The idea has been derived from network standard terminals which have been developed for resource sharing networks, such as Arpanet.

Ultimately it may be possible to remove the virtual-to-physical-terminal conversion, when virtual- and physical-terminal protocols correspond directly. It is therefore important that the virtual-terminal protocol should be specified with the overall architectural concepts of open networking in mind since this is the real-terminal protocol that the world will inherit.

It is also important to note that any differences between the virtual-terminal protocols which are used in separate networks must either be known by the host which is driving them or be handled by conversion at the gateway between the networks and, unless the VTPs in the various networks are harmonised, gateway conversion for all classes of operation will be an almost impossible task. The Inter Network Working Group of the IFIP is carrying out very good work in identifying the VTPs that exist in the various worldwide networks and analysing their differences.

12 CCITT recommendations for packet assembly and disassembly

The CCITT has produced a family of recommendations to cover the attachment of 'teletype'-compatible terminals to networks. These range from conventional 'scroll-

mode' teletypes to very simple screen terminals which communicate in teletype mode. As previously discussed, asynchronous teletypes are usually unbuffered and the need for packet assembly and disassembly (PAD) as a terminal service is introduced. The recommendations define the PAD facilities and how they are controlled by either a remote host or the terminal that is being served by the PAD. They are numbered X3, X28 and X29.

The recommendations were designed ahead of the open-network studies and it has already been recognised that they do not fit neatly into the system architectural model that has been proposed. The recommendations include only basic functions and it is necessary for the mainframe software to be aware of any complex characteristics of the device that is being driven through the PAD and any special display-manipulation facilities that are available.

X3: defines the facilities of the PAD. In addition to the basic packet assembly and the virtual call-control functions of X25, which it performs on the behalf of the terminal, it also includes user-selectable functions such as 'echo', a range of message terminations, idle timer periods, suspension of output, discarding of output, line folding, 'padding' after carriage return etc. The PAD specification also includes parameters that are predefined for each terminal connection, such as the line speed (up to 300 bit/s in the present recommendation).

X28: defines the interface between asynchronous terminals and the PAD. It specifies the call establishment and clearance procedures for the terminal and the interchange of control and service signals. It also describes how the PAD selectable functions are controlled and checked by the terminal and the responses by the PAD. All the control and service information to and from the terminal is transferred by character sequences.

X29: defines how a remote packet-mode terminal (normally a host) communicates with an asynchronous terminal via a PAD and how it may access and control the user-selectable functions in the PAD.

13 Other asynchronous virtual protocols

A draft Euronet Standard for a scroll-mode virtual terminal (ESP25) was close to agreement before the CCITT produced its recommendations X3, X28 and X29 but this has now been discarded since the CCITT recommendations satisfy the Euronet requirements.

Arpanet already has a standard for teletypes and display terminals which operate in scroll mode, known as Telnet. Telnet differs from the CCITT PAD recommendations. The most fundamental difference is that Telnet embeds all control characters in the data stream and identifies them by an escape character, whereas the CCITT recommendation uses separate sequences for control and parameter signalling.

This gives a clear indication that, even at the simple scroll-mode-terminal level a lot of work still has to be done.

14 Synchronous virtual-terminal protocols

CCITT X25 makes adequate provision for the connection of synchronous terminals and the CCITT is not currently working on synchronous virtual-terminal protocols. This will be the responsibility of the ISO in the long term but, because adequate standards do not exist, network implementers are making their own provision for the connection of the existing population of terminals.

Euronet is studying the synchronous-terminal area and, in particular, single and clustered interactive terminal systems and remote job entry. Nearly all existing terminals in these two classes use a derivative of the basic-mode link-control protocol with some differences of interpretation. However, the reconciliation of the differences in link protocol is a small problem compared with those that exist at the higher levels where each manufacturer has introduced very sophisticated facilities.

In video display systems, the accent is on screen-manipulation facilities involving cursor-movement functions, text manipulation (such as 'rack up') and hard-copy features. If a virtual-terminal protocol is produced for this class of terminal, all the facilities that are considered to be desirable in the real terminal of the future must be standardised, including the size and basic geography of the display screen. In the meantime, the mapping between the virtual and the physical terminal will introduce many problems. For example, if the application level in this host processor produces a tabular format for a virtual-terminal standard screen size it will, undoubtedly, produce an illogical format on a screen of a different size, regardless of how clever the mapping function tries to be. At present it is also necessary for the interactive-terminal operator to be aware of the formal dialogue that has to be exchanged with each specific host operating system and no amount of translation by a VTP converter can replace this.

Euronet has produced a screen-mode specification for synchronous interactive terminals known as the data-entry virtual-terminal protocol (DEVTP). Like the scroll-mode VTP the specification defines a basic set of functions but the DEVTP contains more sophisticated elements such as operator interrupt and activation sequences and screen-manipulation facilities. It also specifies a range of screen sizes. Although the current specification will be implemented by Euronet it can only be regarded as a first attempt and improved versions have been proposed that conform more strictly to the architectural model that is being proposed for open system operation.

The same synthesis problem exists with RJE terminals but it is compounded by the range and characteristics of the peripherals on the terminal and its degree of intelligence. In the open-network concept, the RJE system may be either a simple passive input-output terminal or a host system which transfers batch jobs on a resource-sharing basis, although it could be argued that this latter extreme is covered better by the host-to-host protocols, described below.

It has not yet been decided whether Euronet will provide a 'black box' for each style of VTP that is supported or whether it will be up to the user to provide his own conversion to the X25 packet interface. It is assumed that the latter will never be precluded. Whichever is adopted, the basic input/output and display characteristics of the terminal will still be visible to the mainframe software and any sophisticated features at the terminal must be known before advantage can be taken of them.

The above problems serve as an indication that the generation of synchronous virtual-terminal protocols will be very difficult and, if they are produced to cover short-term needs, they may not satisfy the architectural ideals described above. Until the right standards are produced it is probably better for terminal and main-frame suppliers to continue to emulate each other's protocols (mapped on to X25), in order to achieve a satisfactory interconnection, than to try to identify the lowest common denominator of the facilities of the existing systems in a particular class. It is assumed that the generation of the new common DTE/DCE interface which has been mentioned above will simplify the type of connection.

15 Host-to-host protocols

For some years Arpanet has been using a file-transfer protocol (FTP) which includes facilities for full access control and identification of the file name in a standardised form. Arpanet has developed the idea of carrying control information on a separate logical channel from that used for data and this requires careful examination against the architectural ideas mentioned above.

Arpanet also has a generalised host-to-host 'interprocess' communication protocol, known as NCP, that simply conveys control information in the leader field of each data message, but a new version is now being tested called 'transmission-control protocol' (TCP) which, like FTP, uses separate logical channels for control and data.

The EIN network implementers have specified a Bulk Function Transfer (BFT) which is intended for more general host-to-host transfer and which can also be applied to file transfer. EIN has copied the idea of using separate logical channels for control and data but there is no direct compatibility between Arpanet FTP/TCP and the EIN BFT and mapping between them will be difficult.

Considerable work is needed in the area of file transfer and interprocess communication for general compatible standards and this work is only just beginning.

16 New work on international standards

Fig. 3 shows the international standardisation groups involved in the various aspects of this subject and shows the distinctions and the interrelationships between their work. It also shows some of the important existing standards at their relevant levels. This list is by no means exhaustive.

The work on structured high-level protocols began in the British Standards Institute (BSI) in 1975 as a result of the architectural studies which had already isolated the link-control functions and led to the generation of HDLC. The BSI has been a prime mover within ISO on this subject and has been directly responsible for the establishment of the new ISO/TC97/SC16 high-level-protocol committee.

The ideal structure for high-level protocols, based on the onion-skin architecture, which was proposed by the BSI, has been consolidated by the ISO committee and a provisional model of open-systems architecture has been produced. This separates the user tasks and working data, the activation of the network and the interface to the transport service. Ways of expanding this structure into detailed specifications have already been considered. These extend far beyond the scope of the virtual-terminal protocols to cover the entire problem of task interchange.

Much work has been done by the BSI on task activation and network set-up, including the criteria for establishing and clearing a link between two systems and the parameters of a network-control language. Requirements for flow planning, control and recovery have been studied. The work has so far been concentrated on a command language for four types of information transfer:

Class 1: Short transaction-oriented messages—which do not require either response from the receiver, or the retention of the communication link.

Class 2: Remote job entry—tasks where the function of the protocol is to signal the actions to be taken by batch-processing resources.

Class 3: File transfer—between systems and, on occasion, between different media. The BSI is taking into account the good work which has already been done in this area by the UK Post Office EPSS user group.

Class 4: Interactive tasks—using either simultaneous (duplex) dialogue or very rapid line turnaround involving the retention of the communications link during processing. Interactive task set-up often involves the use of RJE-like protocols to indicate resource requirements, and arrange for file transfers before or during processing.

One possible way that has been considered for distinguishing at the command level between the different classes and different messages in the same class is to use command verbs with different semantics, i.e. put/get for straight message transfer, take/give for RJE operation etc.

The interface to the transport service (Fig. 4) has been considered and the basic functions that should be provided within the transport service have been defined in outline, including the commands which interface the higher levels to the transport service. A standard transport-service interface that is independent of the communication network has been identified.

As these standards mature in the international arena, they will lead to a new dimension in networking. It will be possible to add new application-oriented interchanges within the same basic procedural structure. The need to perform conversions at several architectural levels for each new terminal type which is introduced will be avoided. However, it will be necessary to pay great attention to device characteristics, and the need for knowledge of special sophisticated terminal features at the application level is inescapable, even with this approach.

17 The Value of Standardisation

17.1 Economic

During the next ten years it has been estimated that \$1,000 million will be spent worldwide in the areas considered in the paper by computer users, manufacturers, software houses and the PTTs. Even if standards can save only a tiny percentage in this sum, their economic value is indisputable.

Conversely, if the absence of standards contributes to extra work, such as reprogramming to meet different procedures, or to any increase in confusion, the wastage will be on a grand scale. Most importantly it will be the wastage of the world's most valuable commodity: skilled manpower!

17.2 *Practical*

It is not apparent how any effective degree of open working can be attained without network standards.

Clearly, users and manufacturers will not be able to implement the full range of the new standards overnight and a major aim of network standards will have to be the harmonisation of related equipment and procedures, to make conversion of existing equipment as simple as possible.

17.3 *Commercial*

The major owners of distributed computer systems and the major manufacturers are currently having to provide packages relevant to the most critical functional levels of Fig. 2, from 'command protocols' to 'link control', inclusive. For example, the airlines (SITA, IATA, ARINC), the European banks (SWIFT), and several other commercial networks, for the lack of any alternatives, are being oriented towards their own immediate needs. The generation of international standards at all the levels discussed could prevent further drifting apart and will, hopefully, bring these users more closely together in the future.

18 **Conclusions**

To permit open operation in the computer-network environment, network standards will be needed.

They will be of significant economic value to computer manufacturers and users.

The need is urgent. Commercial organisations are having to provide customised equivalents, but these are not yet so rigid or widely adopted as to prevent modification to attain some degree of harmonisation.

There will be great benefits in looking ahead to known needs, rather than constantly suffering the costs of trying to convert previously established, different schemes.

Acknowledgments

The most significant contribution in this area was the original input on system architecture by David Ackerman of ICL which set the pattern for all this work.

Bibliography

Much of the information that has been described is currently only published in the papers of the relevant standards organisations or the handbooks of the network-implementation groups and is not all freely available. The basic-mode and HDLC link-control standards are available from the BSI under BS 4505 (parts 1 to 7) and BS 5397 (parts 1 and 2), respectively. The CCITT X series recommendations have been published in Vol. VIII of the CCITT plenary records for 1977 (known as the 'orange' books), and a special supplement containing recommendations X3, X25, X28 and X29 is available from the CCITT publications Department.

Distributed computing in business data processing

M.V. Wilkes F.R.S.

Professor of Computer Technology, Computer Laboratory,
University of Cambridge

Abstract

The paper discusses the extent to which developments in minicomputers and in micro-electronics are making it necessary to reconsider the role of a large central computing installation in a business environment.

1. Introduction

In the computer field, as in other fields, we have our OK terms. The current one is distributed computing. Everybody is talking about how minicomputers are becoming more and more powerful and less and less costly, and we keep reading articles about things that the semiconductor industry has up its sleeve. Some people will tell you that the big centralised computer centre is already out of date, and that the work could be done better and more cheaply if every department or office had a minicomputer or two of its own. All that is necessary, they say, is that people should stop being reactionary and get on with it. More sober heads point out that, even if the trend is admitted, you cannot break down an organisation and replace it by a new one overnight, and that there are other factors besides the availability of equipment to be taken into account when comparing the advantages of decentralisation with those of centralisation.

Before one can form a rational view of what is likely to happen and decide how immediate the threat to established interests really is, one has to understand clearly what the technical developments are and what part of a data-processing installation they affect. A data-processing installation consists partly of mechanical equipment and partly of electronic equipment; often the two elements are mixed in the same box. On the whole we are much happier nowadays with the electronic part of our data-processing systems. This was not always the case, and it is a benefit that has come to us as a result of the development of solid-state circuits.

2. Disc files

We would dearly like to see the replacement of the rotating disc file by something

with no moving parts, but this desirable end can hardly be said to be in sight. On present showing, bubble and CCD memories are unlikely to provide the capacity required at reasonable cost. The only approach of which I am aware that might lead to a device with a capacity rivalling that of a disc file involves the use of a steered electron beam writing on a semiconductor target. Even if such a device could be put into production, serious problems of reliability and life might be expected, particularly as semiconductors do not like being bombarded with high-voltage electrons. On the whole, the chance of a nonrotating replacement for the disc file becoming available in the foreseeable future appears remote. One does not, of course, know what is just round the corner; something may turn up, but for my part I shall believe it when I see it.

We may still be dependent on rotating magnetic memories, but the present-day ones are a great advance, both in cost-performance and in reliability, on the primitive devices that we had 20 years ago. Developments in this area may not have the same glamour as developments in semiconductors, but the progress has been truly remarkable. Few of us would have predicted 20 years ago that densities anything like those achieved in modern disc files would be possible, let alone compatible with increasing standards of reliability. Moreover, we have not yet reached the end of what can be achieved. High performance disc files do, however, require regular attention and work best in a dust-free and air-conditioned environment. They take up a lot of space in the computer room, largely because enough space must be left around them for access by operators and maintenance staff. If the wished-for day ever arrives when disc files are replaced by some solid state—or even liquid state—device, not the least benefit will be that we shall be able to put the entire computer in the basement, with no environmental control other than the removal of heat, and forget all about it.

The density at which information can be written on magnetic tape has also increased markedly over the years and is still increasing. As disc files have been improved, however, the role played by magnetic tape has steadily become less important. Nevertheless, magnetic tape is so ideally suited to archival and long-term storage that I see a continuing role for it whatever happens.

3. Semiconductor electronics

While we can certainly expect further useful improvements in efficiency and reliability in mechanical peripherals—including printers—it is obviously in the electronics that the most dramatic changes will occur. After an uncertain start in the late 1950s, transistors rapidly became more reliable and faster in operation. During the last 10 years the use of discrete transistors has been phased out in favour of integrated circuits containing a small number of gates and flip-flops. It is to these developments, rather than to the much more recent, although long foreseen, breakthrough into large-scale integration (LSI), that we owe the low cost of the mini-computer.

LSI is now an accomplished fact and has given us pocket calculators, digital watches, and microprocessors. Microprocessors are to be found in increasing numbers in business data-processing installations, but they are tucked away in disc-file controllers, printers, and the like. Microprocessors have already found their way into the processing units of low-performance minicomputers. Soon they

will be used in the bigger and faster minicomputers as well. This will, however, have only a marginal affect on cost since the dominant element of the processors both in cost and number of chips will be the memory. The real impact of LSI on general-purpose computing will not be felt until we have low-cost memory as well as low-cost microprocessors. It will be some years before this comes about, but sufficient is known of the state of progress in the semiconductor industry for some fairly confident projections to be made.

The cost of the memory depends on the number of bits that can be stored on a single silicon chip. From the point of view of the semiconductor manufacturer, the present state of the art is 16k bits per chip. However, although manufacturers have been delivering chips of this kind for over a year, the number that have found their way into computers that have actually been delivered to customers must be very small. Most computers with semiconductor memory, in fact, have 1k bit chips corresponding to the state of the art of several years back, although the newer models use chips with 4k bits. It is necessary to use 128 of these chips to provide a 16 bit minicomputer with 32k words of memory. With 16k bit chips this number drops to 32 and, when the point is reached at which chips holding 64k bits can be manufactured, only eight chips will be needed. Present indications are that this point will be reached in the early 1980s and if this is so the chips should be in widespread use by the middle of the decade. During the next few years, however, for any worthwhile computing, as distinct from control, application memory costs will dominate and it is important that we should not allow ourselves to be misled by the low cost of the microprocessors themselves.

4. Small and large computers

In spite of the fact that the semiconductor revolution has still some considerable way to go, the minicomputer is well and truly with us and has been for some time. Indeed, the name is fast losing its point. It once meant a computer at the very small end of the range—off the end of the range, perhaps—but now the minicomputer, given sufficient memory and a full range of peripherals including disc files, is indistinguishable from a medium-scale computer. In fact, one may expect that all but a few of the biggest and fastest mainframe computers of the future will be seen as descendents of the modern minicomputer rather than as descendents of the mainframe of today.

Some people take the view that there will, for economic reasons, be no future for the superfast mainframe. The ground is certainly being cut from beneath its feet by the grown-up minicomputers to which I have just referred. These will be produced in quantity and therefore at low cost. The argument is that there will no longer be a cost advantage in using the single superspeed processor and that it will be better to use a number of slower but still fast processors. This may well be true in business data processing and in much scientific computing where it is possible to break the work down into self-contained jobs—indeed, in business data processing this would only reverse the process that originally brought a large number of such jobs together in order to achieve economy of scale. There are some scientific tasks, however, that are indivisible, although no-one seems to know how many. Possibly the superspeed computer will survive for these. Another possibility is that those who have long scientific calculations to perform against a deadline will

commission the building of computers—or rather high-performance processors to be attached as peripherals to computers—specially adapted to their needs. Designing such a processor would be a very much easier task than designing a whole system that would fit into the top end of a range and be suitable for offering as a product on the market. It has escaped the notice of some observers that the building of one-off processors is now becoming an entirely feasible and economic proposition.

The use of a number of minicomputers instead of a single mainframe does not mean necessarily that the minicomputers should be scattered around the organisation. They could perfectly well be grouped together in a central computer room. This would have the advantage, among others, that they could share the peripherals and filing facilities. The new developments may be seen as giving us freedom to concentrate or disperse our computing resources as we may find convenient in given circumstances.

With the present balance between the cost of printers, disc files, high-speed memory, and processing power, applications favouring geographically decentralised processing are those in which little storage, either long-term or high-speed, is necessary and a simple printer for output and a keyboard for input will suffice. These are all stand-alone applications such as may be found in small businesses and in professional offices of various kinds. The alternative that these small local systems tend to displace is a connection to a time-sharing bureau. Like terminals on a time-sharing system, the small computers can be used by the ordinary personnel in the office and it is not necessary to employ anyone in a specific computer role. Once it is necessary to attach a high-performance printer and a large disc file, this ceases to be the case and the employment of computer operators becomes necessary. In fact, before one knows where one is, one can have all the problems of a small computer centre on one's hands. It is then, in a not so small organisation with a number of such centres, that one begins to wonder whether a single centralised computer service would not be better after all.

At the other end of the scale we may consider a large computer centre in a public utility whose main role is invoicing and the issuing of bills to the public. Unless such a centre is grotesquely large, it is not easy to see how economies could be achieved by splitting it into two or three independent centres. More high speed printers would certainly be required in order to provide cover against mechanical breakdown; more storage would be required because of the fragmentation of the spare capacity and because certain information would undoubtedly have to be held in all the centres; air-conditioning and overhead expenditure would almost certainly increase, as would the expenditure on supervisory staff. From the theoretical point of view, the optimum economic size of a computer centre with a clearly defined workload must be quite large and in practice its size is likely to be limited by other considerations, such as security and managerial unwieldiness. What will perhaps come to be questioned is the wisdom of loading the centre up with a great variety of work that has nothing to do with its main role.

5. Access to data

The day will undoubtedly come when there is no longer any economic need for computers to be shared between large numbers of different applications. In many applications, however, a connection to a central filing system will be essential in

order that data may be shared. A connection to a hard copy centre for printing and plotting may also be a convenience.

Filing systems of the kind familiar to Cobol users are the direct descendants of systems that were originally developed in the 1950s to use magnetic tapes and were later adapted to take advantage of the improved access offered by disc files. Over the years the systems in use in large companies have been subject to continual modification and evolution, and the feeling grew up a few years ago that the time had perhaps come to make a clean sweep and to replace the miscellaneous collection of files by a central database. An ideal that was frequently spoken of was to have in a large company a single database that would contain without any duplication all the information required for carrying on its activities; this database would be accessed by an application program written to meet the needs of the individual and diverse departments into which the company was divided. It was said that efficiency would be promoted by bringing all the data together and that the absence of duplication would make inconsistency impossible.

The operation of a corporate database serving many different masters would raise a host of problems and resolving the data bottlenecks that occur would tax the resources of any data administrator. It seems to me that a company going in for one of the modern database management systems will not attempt a simultaneous changeover to a new total system, but will proceed by stages, transferring various activities one at a time. The result is much more likely to be a set of federated databases rather than a monolithic database. The advantage of running some of these on independent machines in order to obtain fully parallel access would soon become apparent. I do not think that the resulting replication of basic data would be in itself a disadvantage, provided that positive steps were taken to check periodically that data items in the various databases that were supposed to be identical were in fact so. The avoidance of replication is only one way of ensuring consistency.

The individual databases in a federated system need not be all in the same location. A factor affecting the decision whether to centralise or not would be the cost of communications. If a federation of geographically distributed databases could be run with a lower expenditure on communications than if they were centralised, this might well be a powerful factor affecting the decision. A lot would depend on the extent to which the sharing of data was really required and the extent to which it was necessary that information scattered around the organisation should be fully up to date; in many instances it might be sufficient for it to be updated at 24 h or longer intervals. It is at least possible that the importance of online sharing of live data between the suborganisations in an ordinary business has been exaggerated.

Nearly all of what has been said above is independent of the type of database management systems used. The design of database management systems continues to be an area in which controversy rages. The Standards Planning and Requirements Committee (SPARC) of the American National Standards Committee on Computers and Information Processing (ANSI/X3) set to work to identify areas in which standards might be formulated. Instead, it found itself engaged in fundamental research on data models. Indeed, much work remains to be done before the underlying problems of database technology are properly understood. Only when this is achieved will a firm foundation exist for the implementation of efficient database management systems.

A general model for integrity control

J.B. Brenner

ICL Product Development Group, Technology Division, Manchester

Abstract

Distributed systems pose a new integrity-control problem: how to achieve consistent processing, file access and recovery with distributed control and local autonomy? The general nondistributed problem is analysed first to yield an understanding of the deep structure of integrity control, applicable to all systems. This is then applied to distributed systems, in the context of the evolving ISO architectural model for 'open system inter-connection'.

1 Introduction

This paper arises from ICL research concerned with loosely coupled and distributed multicomputer systems. The subject of integrity control is a vital issue which is general to all computer systems, but particularly taxing in this environment.

We start with a brief survey of the problem area. This is a set of intractable problems of concurrent access control and recovery, which are the main aspects of integrity control as considered here. The subject matter will be new to some readers: to others it will already be very familiar. What is less generally understood is that the difficulties are now seen to be diverse surface symptoms of a deficiency in the deep structure of systems. For this there is a simple structural solution.

The appropriate structure and its implementation are described. It is general to all systems, and is also readily integrated into distributed systems and their protocol structures.

The ideas presented here have been taking shape for several years in the database field. They are now beginning to appear in DBMS products, e.g. the latest ICL 2900 IDMS,¹ but they have only recently crystallised into a coherent and general model for integrity control.

Many different researchers are working on this same problem area, and have reached essentially the same conclusions at about the same time. Another convergent factor is the application of these concepts to standardised 'open system inter-connection', which is being studied by the International Standards Organisation (ISO TC 97/SC 16), and by participating bodies such as the British Standards Institute (BSI) and the American National Standards Institute (ANSI). The paper also draws on this work, in which ICL is actively involved.

2 Survey of the problem area

2.1 *Problem outline*

The purpose of this section is to provide an agreed summary of the state of the art and its difficulties before breaking any new ground. We are concerned here with three different but related problems:

- concurrent access control
- data recovery after failure
- process recovery after failure.

Each problem is first introduced individually and then the crucial interrelationships are described. Finally, we consider why the associated difficulties are particularly acute in loosely coupled systems.

The description is mainly in terms of file data and its processing. This is the most familiar manifestation of the problems. From it we can correctly generalise to all kinds of processing involving shared resources.

2.2 *Concurrent access control*

The essence of the concurrent access control problem is:

- whenever otherwise correct and separate processes share access to the same data, chance data interactions can cause errors, e.g. inconsistent inputs and double/lost updates.

Concurrent access to shared data is a general requirement, therefore these problems must be overcome.

The general approach is to ensure that each process gains temporary and localised exclusive or protected access to data. This confinement allows detection of the chance collisions between concurrent processes. Errors that would otherwise occur can then be prevented.

The usual means of achieving confinement is by file assignment, and then finer-grained data-locking techniques. These prevent other processes from reading and/or writing the particular data concerned. The techniques are well known, well understood, but often poorly implemented.

Allocation and locking entail numerous technical problems. Some of the most important are:

- what to lock—*which data, messages etc?*
- which lock granularity—*bit, record, block, set, predicate etc?*
- when to lock and unlock these entities?
- which kind of lock—*wholly exclusive, prevent writes, prevent reads?*
- who does locking—*applications program, DBMS, operating system?*
- how to handle consequent detected collisions between processes?
- how to recognise and handle the special collision case of deadlock?
- usability problems of associated language interface

- insidious errors. Difficult to create conditions during testing, difficult to detect and contain operational errors, and to identify and fix their cause
- performance overheads. Allocation/locking can involve extensive and intricate actions, degrading response times and consuming resources.
- false collisions between processes can cause a high proportion of failures and reruns when there is intensive sharing. Such errors are usually due to indiscriminate or coarse-grained locking.

Each problem has many solutions, and there is a corresponding diversity of solutions to the complete set. Even within the same system, different parts (e.g. different applications, programs and packages, the DBMS, and various aspects of the operating system) tend to use different techniques. The resultant implementations are usually complex, and often demonstrably lacking in integrity.

2.3 *Data recovery*

Systems generally need reliable data storage. The requirement is:

- prompt dependable detection of failures, tight confinement of consequent data errors, and efficient dependable recovery to a correct, consistent and acceptably up-to-date state, from which use of the data can resume.

Techniques for the data-recovery aspect of this are well known. They include:

- transient data and files that automatically vanish after failure, and by their complete absence restore consistency
- fallback to previous file generations
- spare copies of current files/volumes (static data redundancy)
- multicopy files, updated continuously in parallel (dynamic data redundancy)
- systematic copying of files to dumps (periodic data redundancy)
- continuous logging of the systems state to recovery journals
- logging of new data to journals when updates are written (after-looks)
- restore files after catastrophic failure by using previously secured dumps or an older generation
- roll-forward from a restored dump to the most recent acceptable correct state, by re-applying the after-looks secured since the dump, or by applications rerun
- logging of old data to recovery journals when updates are written (before-looks).
- roll-back after localised failures to re-establish the most recent acceptable correct state, by using the recent before-looks to undo updates.

Actual implementations vary in scope, details of technique and effectiveness, but there are now few areas of technical uncertainty. Understandably, there are performance trade-off difficulties, owing to the overheads of dumps, journals and multicopy data redundancy. As we see later, the crucial technical problems are now all at the periphery, where data recovery integrates with process recovery and concurrent-access control.

2.4 *Process Recovery*

The requirement is similar to that for data:

- prompt dependable detection of failures (but also a deliberate abandon facility), with tight confinement of consequent process errors, and efficient dependable recovery to a correct, consistent and acceptably up-to-date state, from which the process can resume.

Techniques for process recovery are well known. The two main lines of approach are:

- discard all results so far, rerun from the 'beginning' (however defined)
- secure designated intermediate states of a process (checkpoints), and restore and restart at the most recent suitable one.

The provisions range from completely automatic to unassisted user-provided recovery. There are now few areas of technical uncertainty. Once again, the main problems are performance overheads, and integration at the periphery with data-recovery and concurrent-access control.

2.5 *The interrelationships*

Data recovery and process recovery are naturally related:

- when a process fails or abandons, this usually requires any data changes made by it to be undone. Both data and process must then recover
- when data fails, this likewise usually requires any process accessing it to fail, and both must recover
- the states restored by process recovery and data recovery must be mutually consistent
- there is a technical similarity. Process recovery is essentially the restoration of some previous state of process memory. This can be viewed and implemented as a particular case of general data-recovery provisions.

The relationships with concurrent access control are more complex:

- the concurrent-access-control system is involved in identifying the above relationships between failing processes and failing data
- unsuitable integration with concurrent-access-control provisions may prevent correct recovery (see below)
- recovery provides a universal escape route from concurrent-access collisions, especially deadlocks
- the implementation of concurrent-access control must itself store data (tables, lock lists etc.). Therefore it is dependent upon data recovery for their correct storage and restoration across failure situations
- vice versa, the recovery implementation must access shared tables and shared data files concurrent with their normal use (e.g. to dump, or for localised rollback).

It is of crucial importance to understand a point made above, that dependable recovery is impossible without correct integration with concurrent-access control. Circumstances in which a process or data might not be recoverable to a previous state are:

- if any resources allocated to the process might since have been released, but may be needed for the restored previous state
- if any inputs which it uses might since have been changed by other processes
- if any of its outputs might already have been used by other processes
- if any of its recovery journal information might already have been discarded.

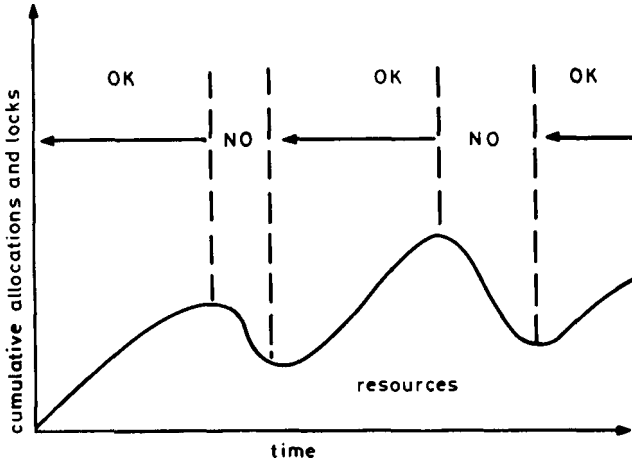


Fig. 1 Profile illustrating that backwards recovery is not generally possible across a period in which resources have been released.

These constraints generalise to the form illustrated in Fig. 1. Moreover, this is not just a matter of consistent behaviour within the one individual process concerned. Recoverability and concurrent-access integrity are also dependent upon the behaviour of any other processes which might chance to access the same data. Therefore the whole population of *all* processes in a system must be subject to *one* universally consistent integrity-control scheme.

Despite the need for accurate consistent integration of integrity control, its provisions tend to be poorly integrated in most state-of-the-art systems, even disjoint. Consequently, they are difficult to use in the necessary coherent manner, and this generally requires extensive error-prone and *ad-hoc* user involvement. With hindsight, it is easy to see what has happened. As shown above, there are intricate and reflexive relationships between the three aspects of integrity control, between them and the individual user applications, and among the many user applications. This readily degenerates into complex convoluted situations. However, given an appropriate structure, it ought to be possible instead to turn this reflexiveness into recursion, which is a source of orderliness and simplicity.

Our analysis also clearly shows that the main problems are at the periphery,

where the three aspects of integrity control and the user processes meet. Therefore the decisive step to rationalise the situation must be to agree on one common unit of user processing relating to all aspects of integrity control.

2.6 Problems particular to loosely coupled systems

Integrity control is essentially involved in co-ordination. Therefore it is particularly significant to loosely coupled systems (e.g. open and distributed systems).

The related technical difficulties are unusual because present techniques and experience are almost wholly based upon centrally co-ordinated and tightly coupled computer systems:

- executive/operating system in a single central computer (or tightly coupled multi-processor)
- central DBMS/filing system providing the sole authority for data-access control, locking and journalising
- communications and terminal systems with an equivalent centralised star structure.

In loosely coupled systems, a more natural model is for there to be many work-stations/computers, generally of equivalent (peer) status, with control and authority somehow distributed among them. Distributed control can also help avoid performance bottlenecks, and critical reliability dependencies. Further, it simplifies effective exploitation of the natural marginal redundancy which is latent in multiple computers.

Until the recent developments, which are the subject of the rest of this paper, there was no general solution to the technical problems of co-ordinating distributed processing and distributed databases. There appeared to be an unavoidable necessity for some central all-knowing hierarchy of intelligence to co-ordinate activity in a distributed system. This is all the more disturbing, because it would seem to imply some deep-seated inconsistency between practicable computer technology and the social and business needs which it must serve. This particular inconsistency would vitally affect social issues such as autonomy and information privacy.

It is now realised that the state-of-the-art technical difficulty with integrity control in loosely coupled systems is not intrinsic to them. Instead, this more demanding environment can be seen as representative of the *general case*, which reveals a structural inadequacy, already present and habitually patched over by *ad hoc* solutions and by resort to the *special case* of central hierarchical control.

This completes the state-of-the-art summary. We are now ready to consider the new findings.

3 Structure for integrity control

3.1 Main concepts

Recent research has yielded an understanding of the principles of a complete and general structure for integrity control. It is quite simple.

The central idea is that all processing can be subdivided into distinct

integrity-control steps, which are called commitment units. These provide a universal granularity for integrity control which is decoupled from the variety and complexity of application characteristics. The nature of a commitment unit is such that if the correct integrity-control behaviour of *each* is individually assured, then that of *all* is *collectively* assured without further provision. This is a vital simplification.

A related idea is that each and every object (resource) can be accessed only through dialogue with its one particular custodian, which has a local responsibility for access control and recovery. Each custodian is usually responsible for many objects, usually in its same locality. Commitment units, and the custodians which they access, interact systematically, such that the correct integrity-control behaviour of *each* individual custodian assures that of *all* custodians collectively. This is another vital simplification.

Further, the synchronisation characteristics of commitment units are suitable for implementation across a loosely coupled system. Also the integrity-control implementation is wholly in software, which can itself be recursively subject to its own provisions. This enables complete and *distributed* integrity control in a distributed system.

From these premises it logically follows that arbitrary and continually changing collections of individual commitment units and individual custodians can achieve correct *collective* integrity-control behaviour. This is essential for true open-system interconnection, because this necessarily entails a continually changing population of participating resources and activities. This contrived amorphousness is also essential for resilient systems which must survive dismemberment.

The orderliness of the integrity-control structure allows adaptation and tuning to minimise related performance overheads. There is selectivity of locking and journalising, and localisation of related traffic within a distributed system. The user has a simple interface to integrity control: most of the provisions are automatic and implicit in the structure. This enhances usability and dependability.

The main components of this integrity-control structure are now considered in more detail:

- commitment units
- automatic integrity control
- custodians
- commitment-unit synchronisation
- recursion

Afterwards, the resultant implementation characteristics are further explored to complete the justification of the assertions made above. The description is in terms of distributed control in a distributed system. This is regarded as the general case. The same principles are applicable to the special case of centralised systems.

3.2 Commitment unit

Each user process defines its own commitment units, which can therefore properly reflect applications characteristics.

The only essential criterion is that a commitment unit should comprise a complete transformation from the generally prevailing stable and consistent state at

its beginning, to a new stable and consistent state at its end. In this context 'stable' means a point suitable for recovery, and 'consistent' means locks are not needed to constrain access. It is desirable that commitment-unit duration and work content are reasonably convenient for concurrent-access control and recovery purposes. That is all. Given these simple characteristics, integrity control can be assured by formal rules, automatically implemented.

The observations in the next few paragraphs are intended to show that this subdivision of processing into commitment units is readily attained.

Transaction processing (TP) is the most clear-cut case. The user's transactions are generally commitment units. Each is a brief but complete transformation to a new stable and consistent state. This is so of all aspects: at the terminal (from before initial input, via wait, to after display of final result); in the communications and processing system (from a null state, via activity, to a null state); in the database (from prior state, via possibly inconsistent intermediate states, to a stable and consistent state afterwards). The integrity-control requirements are that during a transaction (i.e. commitment unit) there should be isolation from the chance effects of others, and that failure or abandonment should lead to restoration of the prior consistent state. Afterwards, the committed durable results should persist, even if there is subsequent failure.

Multi-access computing (MAC) sessions are likewise readily broken down into transactions and other distinct subdivisions. These commitment units have the same kind of integrity-control requirements as in transaction processing.

Batch processing tends to be a concatenation of what might otherwise be separate commitment units. Alternatively, it can be viewed as successive read and update cycles, file passes or job steps. These would provide rather larger commitment units. However, the integrity-control requirement is essentially the same in all cases: during a commitment unit there should be isolation from chance interactions with others, and until its end the outputs should generally be uncommitted and able to roll back if in error. The end of each such commitment unit provides a secure committed starting point for any direct successor.

In a busy system, there will be many concurrent commitment units, of varying duration (Fig. 2). The concept 'commitment unit' has previously been referred to as a 'database transaction',² and a 'success unit'.³

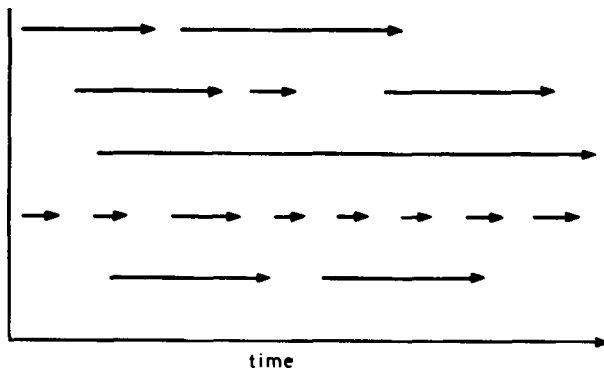


Fig. 2 Multiple concurrent series of commitment units, each distinct and asynchronous

The commitment unit concept does not exclude the possibility of other access control and recovery granularity. Generally there will also be finer-grained recovery and retry steps within commitment units and associated transport services. However, the commitment unit is absolutely fundamental to the structure, because it is the one place where user processing, concurrent-access control and recovery all meet and integrate systematically.

3.3 *Automatic integrity control*

The general nature of the requirement should already be fairly well apparent. There is a formal definition of the rules for integrity control as 'degrees of consistency' in Reference 2. The necessary behaviour of commitment units is described here more informally. It is later explained how this is largely delegated to the custodians involved.

To achieve distinctness of each commitment unit:

- any value input to a commitment unit is then automatically locked until its end, to prevent others from altering it (defined exceptions)
- any value output by a commitment unit is then automatically locked until its end, as uncommitted and inaccessible to other (defined exceptions)
- all commitment units are automatically constrained to respect these current locks (some special exceptions).

This automatic distinctness (i.e. isolation) and deferred commitment not only assures concurrent-access integrity. It also means that if a commitment unit fails or is abandoned, its potentially erroneous outputs are already confined, and errors will not have propagated.

To provide the automatic means of current commitment unit rollback and restart:

- before the beginning of a commitment unit (usually at the end of its direct predecessor) a check point (or equivalent) is automatically taken and secured (no exceptions)
- before each uncommitted output is recorded, a before-look (or equivalent) is automatically taken and secured (no exceptions)
- the automatic locks prevent these uncommitted outputs from being overwritten by others (no exceptions)
- a commitment unit is automatically prevented from releasing allocations or locks before its end (defined exceptions)
- the automatically recorded journal entries supporting commitment-unit recovery are not discarded until after its end (no exceptions).

Finally, to ensure that the durable outputs of successfully completed commitment units can persist, even if there is subsequent failure:

- all durable outputs of a commitment unit are automatically secured in non-

volatile memory, and are subject to automatic after-look journalising (or equivalent, no exceptions).

The exceptions to the automatic provisions (mentioned above in brackets) are discussed later. They are intended to allow flexibility and selective minimisation of associated performance overheads, but without weakening overall integrity control.

The combined effect of these automatic actions is that the tangible effects of a commitment unit are generally deferred until its end. At this singular point, the changes become committed, and generally accessible. Rollback is then no longer feasible, if only because outputs may have propagated elsewhere and cannot be snatched back (except with complications which compromise generality).

3.4 *Custodians*

The role of a custodian has a direct and beneficial equivalence to the structured programming concepts of monitors⁴ and information-hiding modularity⁵.

Each individual custodian is responsible for autonomous integrity control for objects in its custody. The two main functions are:

- current local-access control
- current local recovery

It is implicit in the structure already described that a custodian can derive the necessary information from the allocation messages, data-access messages, and commitment-unit end and restart messages which it receives. A custodian would usually also administer related security/privacy arrangements.

To achieve its required integrity-control function, a custodian autonomously maintains allocation details and locks which define current relationships between objects in its custody and commitment units. It will reject any message not consistent with these. It also helps maintain related recovery information, such that a locality can dependably provide local rollback and restart for current commitment units known to it. This is done by means of before-looks, and by securing the local outputs and perpetuated allocations at commitment-unit end (equivalent to local checkpoint for successor commitment unit).

Each individual custodian can therefore ensure that data in its custody is stable and consistent, even if accessed by many current commitment units (distributed access). The global 'instant' of each commitment unit end (preceded either by its normal completion or complete rollback), ensures that its outputs are only visible when they are globally consistent. Therefore data that is fragmented and stored in different localities (distributed data) can be kept consistent by the collective effects of the individual distinct commitment units and custodian processes involved.

Consistent dynamic update of multiple redundant copies of the same data in different localities requires no additional provisions. The custodian processes individually behave normally. Again the commitment-unit end synchronising 'instants' ensure global consistency.

3.5 Commitment unit synchronisation

In principle, each commitment unit should have an indivisible end 'instant'. Indivisibility is a general requirement for synchronising mechanisms. But this particular need to be instantaneous is rather special. In the example profile in Fig. 1, dependable rollback is not generally possible if assignments and locks have already been released. It therefore follows that, to be a dependable recovery unit with no indeterminate unrecoverable states in between, the equivalent profile must be like that illustrated in Fig. 3. The domain of allocations and locks attributed to a commitment unit may only decrease by instantaneous collapse at its end.

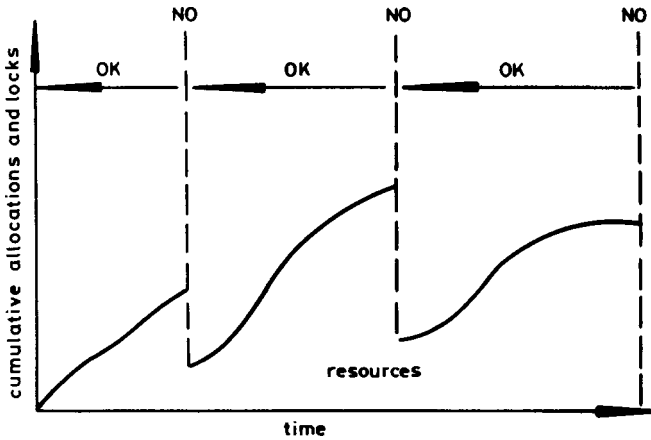


Fig. 3 Profile illustrating that complete dependable rollback coverage is possible if resources are only released at defined 'instants'. Rollback through such 'instants' is not generally possible

In practice, this is pseudo-instantaneous, because it can entail a significant amount of work distributed in time and place, with potentially many participants (all the computers, work stations, processes and services involved.) Its implementation must also be fail-safe and reasonably efficient.

Therefore a careful automatic sequence of actions is used to effect the end of a commitment unit (and the beginning of any direct successor). This is illustrated in Fig. 4.

The securing phase (1) checks with all processes involved (e.g. custodians) that they are alive and well, and that the commitment unit can safely end. Before responding, each custodian will store in nonvolatile memory relevant durable outputs, and the starting conditions of the direct successor (if any). Fig. 5 illustrates the flow of messages.

The master commit phase (2) records in a journal that the commitment unit is now committed to end. The actual (disc) write serves as an indivisible instant, after which the commitment unit is committed and cannot rollback. If there is now failure before the commitment unit is properly ended, recovery discovers this write and can restore durable outputs, finish the ending sequence, and restart any successor commitment unit.

The general commit phase (3) propagates the ultimate end 'instant' by messages

to all participants. The flow is the same as in Fig. 5. Because rollback is no longer needed (continuity by restore and roll-forward instead), the collapse of allocations and locks can be dispersed and of variable duration, but without violating the principles behind the general model in Fig. 3.

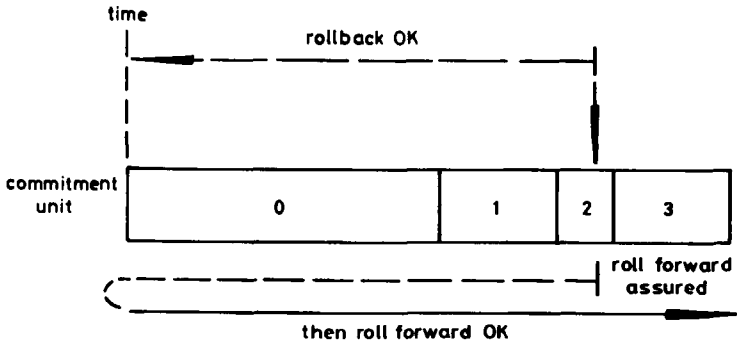


Fig. 4 Progressive ending of a commitment unit and its integration with recovery
 The phases are:
 0 = processing
 1 = securing
 2 = master commit
 3 = general commit

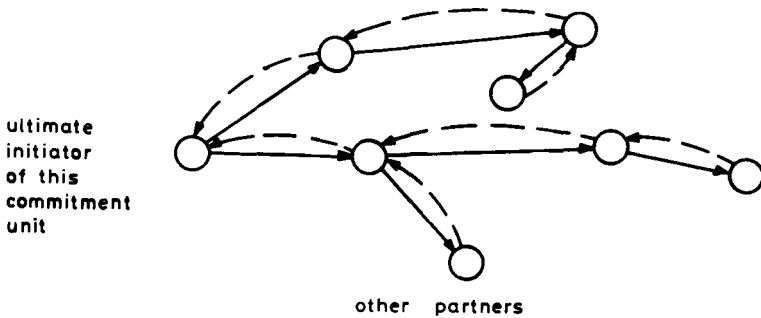


Fig. 5 Propagation of commitment unit ending phase messages in all dependant sessions to all custodians involved and complete confirmation of the actions of all

As each custodian becomes individually aware of the end 'instant' (receives message), it automatically takes the necessary actions within its domain on behalf of the ended commitment unit:

- passes on the news to any relevant dependent process/session
- commits its (the commitment unit's) uncommitted outputs
- frees its automatic data locks (or equivalent transient assignments)
- actions any deferred release of its assignments
- frees related local-integrity-journal space
- makes the immediate successor commitment unit (if any) locally fully current. This generalises to a universal means of notifying the beginning

of a commitment unit (if no actual predecessor, end of null one)

- prepares local-integrity-journal entry to confirm all this (actual write can be later to assist performance)
- responds affirmative to message sender after this has been done and any dependent processes/sessions have all responded affirmative.

Any user-defined data structures can contain local/global commitment-unit identifiers. By cross reference to a local table of locally current commitment-unit identifiers, such data structures can achieve an 'instantaneous' change of state, synchronised indivisibly to commitment-unit boundaries. This also gives a self-tidying effect after commitment-unit failure, and is the recommended technique for all data structures used to implement the above automatic systems functions (Fig. 6).

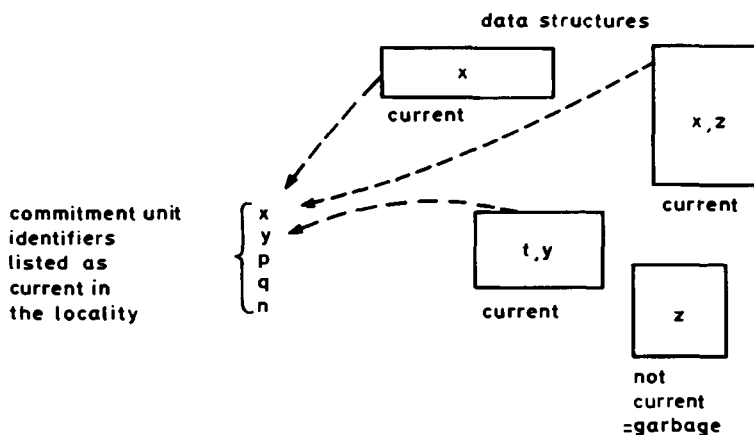


Fig. 6 Data structures with currency synchronised indivisibly to commitment-unit boundaries by quoting commitment-unit identifiers

It is apparent that this one kind of synchronising 'instants' (commitment-unit end) can provide general co-ordination and *all* the intra- and interprocess synchronisation needed for integrity control.

In a centralised hierarchic control system, universal perception of 'instants' is intrinsic (e.g. CPU interrupt, central table update). Such readily contrived 'instants' tend to proliferate, with complex integrity-control ramifications, and inbuilt dependence on centrality. The new model carefully avoids this. Moreover, in a loosely coupled system, synchronising 'instants' involves considerable overheads, and must be contrived with particular care if race conditions and other reliability hazards are to be avoided.

3.6 Recursion

The concept of recursion here is simply that the software and data structures used to implement integrity control should themselves be subject to its own provisions. This is technically easy to arrange if the need is properly foreseen.

This has the immediate benefits of simple resolution of the intricate interrelationships of the various aspects of integrity control, and of making its coverage complete. Proper integrity control should be universally effective: nothing should be exempt. Because the proposed techniques can be implemented entirely in software, this completeness of integrity is attainable by recursion.

Recursion also means that the implementation can be enriched by using some of the more advanced facilities which are implemented. Two such are:

- coherent distributed processing of distributed data
- multicopy data consistency.

In this way the integrity-control implementation can itself be made fully distributed and highly resilient.

We can generalise from this that control and operating systems (as well as applications systems) could also be distributed and highly resilient. 'Meta data' such as catalogues, directories, schedules, journals and DBMS schemas and their associated functions could be distributed, subject only to performance constraints (which can be minimised by careful placement and suitable intercommunication bandwidth).

4 Implementation characteristics

4.1 *Concurrent-access integrity*

The general structure and its implementation have already been described. Exceptions and deadlock are now considered.

Locks are automatically formed by a custodian when objects in its custody are accessed. In general the scope of each individual access implicitly defines the object to be locked (e.g. address, data record, block, set, predicate). In principle, all objects accessed are locked, and in ways implicit in the mode of access (read, write, etc.). However, in practice, some relaxation is necessary for flexibility and efficiency. Allocations limit the permissible ways in which accesses may qualify their locking effects, and can also frequently obviate the need for locking (e.g. exclusive or protected allocation automatically inhibits locking).

The exceptions considered here are:

- unlocked inputs
- write without prior locking
- uncommitted input
- immediately committed output
- orthogonal sanctions.

There are many applications that can legitimately dispense with the automatic (not overwrite) locking of most if not all of their inputs. This reduces overheads and the likelihood of unnecessary collisions between processes sharing the same data. Also there are some inputs that are dynamic and inherently unlockable (e.g. sensors and time inputs).

If unlocked data are subsequently updated, it is then necessary for the commitment unit to prove to the custodian process that it knows the current value that its

update would overwrite (to avoid double/lost updates). This can be implemented by quoting either the existing value in full, or perhaps a more compact checksum, or by a version-number convention. If mismatch, then the update is rejected, and the commitment unit must recover to a point where it can reread and then resubmit an alternative update. Such rejection is generally likely to be infrequent, and should not be disruptive or onerous when proper recovery facilities are provided. This technique also avoids failures caused by locks attributed to delayed or failed/sleepy processes.

Occasionally, systems functions (and more rarely user applications) have a legitimate need to read uncommitted outputs of current commitment units. Such a 'dirty-read' facility could compromise integrity control, so should be highly exceptional. The preferred alternative is to rationalise the situation by having used an immediately committed write mode. Some examples of outputs which must be immediately committed and visible, and impossible to roll back are: outputs to recovery journals, and certain mechanical and electrical outputs.

Sometimes locking effects are required which are not implicit in the actual accesses or the unfolding assignments. An example is the subtle locking of nonexistent 'phantoms' in Reference 6. All such exceptions can be covered in the general model by a facility which we have called orthogonal sanctions. A commitment unit utters a sanction to a custodian. This sanction is an indirect reference to an implementation-defined data structure somewhere which somehow specifies the constraint. The custodian need only ensure that all other commitment units accessing the object concerned are aware of the most recent sanction reference. The user-data structure specifying sanctions should be synchronised to commitment-unit boundaries, using the general technique already illustrated in Fig. 6.

Moving on to another topic, *deadlock* is readily handled. The orderliness and completeness of commitment-unit structure facilitates deadlock detection (all allocations, locks and events are attributable to commitment units, for which there is some kind of unambiguous identification scheme). Any commitment unit blocked by another should check for deadlock. This is implemented by dialogue among (or on behalf of) waiting commitment units, to discover if any of their interdependencies are circular.

If there is deadlock, then rollback by any one of the commitment units involved provides a simple, safe and certain escape route. Complex deadlock-avoidance strategies are unlikely to be necessary or worthwhile, but preclaiming of resources early in a commitment unit may be a useful tactic for applications to use selectively. This should minimise the frequency of deadlock/rollback and associated overhead costs, because deadlock can only occur while the domain of assignments and locks is expanding. Fig. 7 is further idealisation of the profile in Fig. 3.

4.2 Recovery

Some refinements of recovery structure and implementation are now described.

Each custodian is responsible for rollback and restart of current commitment-unit effects within its domain. Convenient implementation involves a local integrity journal, stored within the locality, and shared by all processes in that locality. This could be implemented as a cyclically re-used direct-access nonvolatile file space (disc or magnetic bubble).

Therefore each locality has autonomous ability to secure (i.e. check-point) the local end/beginning conditions of all commitment units known to it, and to roll-back selectively their local effects. This local autonomy improves the efficiency of recovery provisions, and enhances general resilience. Collectively, these local recovery provisions, synchronised to commitment-unit boundaries, can provide complete global rollback/restart coverage. Recovery from various failure situations is now considered.

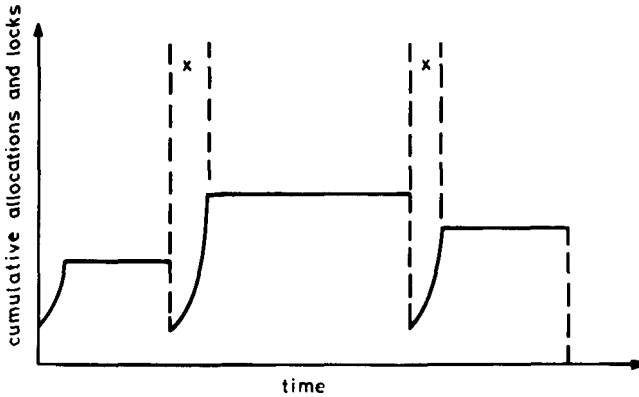


Fig. 7 Profile further idealised from Fig. 3 so that deadlock can only occur in brief periods X

Commitment-unit failure: The commitment-unit originator tells all localities/custodians concerned to rollback that particular commitment unit, then ends and restarts it. Alternatively, this is initiated by some watchdog activity which discovers that a commitment unit has failed.

Locality failure: All commitment units current in the locality are spontaneously broken and rolled back when the locality recovers. The survival of the local-integrity-journal information makes this possible. Even if more direct provisions are absent, the automatic commitment-unit end sequence necessarily discovers locality failure or its consequent spontaneous roll back. Therefore roll back, ending and restart of the affected commitment units is ultimately global.

Concurrent-access integrity is automatically assured during recovery, because within each custodian, the allocations and locks of a rolled-back commitment unit remain current until it is ended in the usual way. Alternatively, if failure destroys the lock lists (or equivalent) in a locality, then all commitment units in that locality with shared data access will be spontaneously rolled back. Their roll-back effects are naturally distinct, and afterwards the lock list is legitimately null, and access to shared objects can resume.

General failure: As for individual locality failure, all affected localities recover and roll back the locally current (and therefore broken) commitment units. Further necessary roll back will be triggered by surviving/already recovered localities. All broken commitment units are therefore automatically recovered and restarted, while those not affected continue normally. It would be recommended practice to

implement start-of-day initial load using this same general recovery mechanism.

Catastrophic failure: The above cases assume that localities survive failure and are able to recover, and that data is not lost, damaged or otherwise unavailable. In the alternative situation of catastrophic failure, data must be reconstructed by resort to duplicates, or dumps and after-looks. These should be maintained in localities with independent failure modes. Therefore the general model need not assume high resilience within a locality. This is an important simplification.

Data written to dumps and after-look journals will retain commitment-unit identities, and include relevant commitment-unit end/begin entries. These provide synchronisation, with the main benefit that at the end of roll-forward, any unfinished commitment units are automatically identifiable. Normal short-term automatic before-looks, taken dynamically during roll-forward by custodians, can be used to roll back and end the effect of these incomplete/broken commitment units. In this way, the restored data can be re-aligned to any chosen consistent state. There is no need for extra predefined special global 'sync. points' or inconvenient global quiescence points during normal running (generally necessary for state-of-the-art recovery systems, and actually impracticable in a widely distributed or truly open systems). Also the overhead of long-term storage of before-looks is avoided.

Multicopy data: If any copy of a multicopy redundant file fails, then the simplest recovery action is for all commitment units accessing any copies to be broken and rolled back to restore consistency. If the failed file is immediately recovered, then processing restarts using all copies as before. If a copy has failed catastrophically, then processing can resume normally using the surviving copy (or copies). At the earliest possible opportunity, full redundancy should be restored by copying a survivor. For this to be convenient, small (subdivided) files are preferable.

Finer-grained recovery: Finally there is the topic of finer-grained recovery to provide partial rollback within commitment units. In principle, it would be possible to define arbitrary mini-recovery blocks within a commitment unit, nestable to arbitrary depth. An example of how useful this would be is implementation of a DML statement (data manipulation language) within such a mini-recovery block. If a part-executed statement (possibly involving many data accesses) is discovered to be erroneous, then these effects could be undone by mini-recovery, before rejecting the DML statement at its ultimate user interface.

In practice there are serious snags. First correct implementation would probably involve elaborate global synchronisation at these minirecovery block boundaries, similar to that for commitment units. This means high overheads. Secondly, this might be technically difficult or even impossible because of conflicts with the basic profile concepts in Fig. 3. Thirdly, the associated major increase of complexity within the integrity-control implementation might impair its *intrinsic* reliability and dependability.

A compromise solution which escapes from this dilemma is described later. Normal re-try mechanisms and related transport-service features (e.g. quarantine units) present no special difficulties. In general because efficient global commitment-unit recovery is provided, the best and simplest response to any failure is always to break, then roll back and restart.

This structure also provides a basic framework on which to implement ultra-reliable software redundancy: dual-coded distinct software,^{7, 8} and recovery blocks⁹. It is now being used for high-reliability database systems.¹⁰

4.3 Protocol language

We assume a layered inter-process communications protocol structure, of the general form illustrated in Fig. 8. This is actually per the ISO TC97/SC16 provisional model for open-system interconnection. (For intercommunication within a locality, the lower four layers would collapse to zero functions).

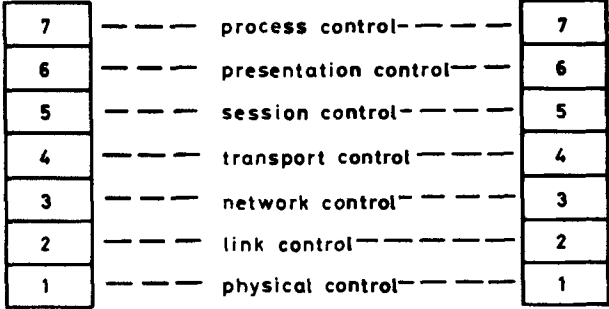


Fig. 8 Seven protocol layers of the provisional model for open system interconnection. ISO/TC97/SC16 March 1978

Integrity control is driven from the highest level. Here the nature of the work defines where commitment-unit boundaries should be. Most of the automatic implementation of interprocess integrity control would be the responsibility of session control (level 5 in the ISO model).

All that the end-user need essentially do for integrity control is to identify begin/end boundaries, and to be able to utter restart requests. The units to be specified are: sessions, successive commitment units within a session, perhaps mini-recovery blocks within commitment units, and quarantine units for transport.

A block structure is clearly appropriate. For example:

- Begin session
 - Begin commitment unit
 - End commitment unit
 - Begin commitment unit
 - End commitment unit
- End session

There is some redundancy in this example. The items bracketed together are indivisible. A condensed utterance may be preferable. At lower levels, not necessarily visible to end users these utterances would result in automatic sequences of dialogue, e.g. 'begin session' (dialogue = initiate and accept), 'end commitment unit' (dialogue = secure, confirm secured, write to journal, confirm, end, confirm).

To avoid the difficulties referred to earlier, it is suggested that minirecovery-block implementation (if any) should be exclusively within the highest level.

process control. It could then be an applications feature, completely decoupled from the general integrity-control model. The ultimate simplification would be for this structure only to yield finer-grained diagnostics (which might then steer user restart), while still using full commitment unit rollback and restart.

5 Conclusion

The deep structure for integrity control in computer systems has been introduced. This comprises a few simple concepts, of which the main one is the commitment unit.

Techniques for automatic concurrent-access control and recovery based upon this have been described here in sufficient detail to demonstrate the possibility of fully comprehensive distributed integrity control. This is generally applicable, and suitable for distributed systems and open-system interconnection. This integrity control provides some of the previously missing basic technology needed to implement distributed databases, and for reliability enhancement by systematic redundancy of hardware, software and data, and their resilient distributed control and co-ordination. The spinoff may also benefit computer security.

This integrity-control structure provides a systematic basis for related standardisation, in which context, knowledge of it is now being rapidly disseminated and generally assimilated into systems architectures.

References

- 1 ICL Manual TP 644 'IDMS Implementation'.
- 2 GRAY, J.N., LORIE, R.A., PUTZOLU, G.R., and TRAIGER, I.L.: 'Granularity and degrees of consistency in a shared database' in *Modelling in database management systems* (North Holland, 1976)
- 3 TOZER, E.E.: 'Preservation of consistency in Codasyl-type database' in *database technology* ISBN 0 903 796074
- 4 HOARE, C.A.R.: 'Monitors - an operating system structuring concept', *Commun. ACM*, 1974, 17, pp. 549-557
- 5 PARNAS, D.L.: 'On the criteria to be used in decomposing systems into modules', *ibid.*, 1972, 15 pp. 1053-1058
- 6 ESWAREN, K.W., GRAY, J.N., LORIE, R.A., TRAIGER, I.L.: 'The notion of consistency and predicate locks in a database system' *ibid.*, 1976, 19, (11)
- 7 GILB, T.: 'Distinct software; a redundancy technique for reliable software'. Infotech State of the Art Report: Software Reliability, 1977.
- 8 FISCHER, FIRSCHEIN and DREW: 'Distinct software: an approach to reliable computing'. Japan - USA Conference Proceedings, 1975
- 9 RANDELL, B.: 'Systems structure for software fault - tolerance' *IEEE Trans SE-11*, 1975, pp. 220-232
- 10 GRAY, J.N.: 'Notes on data base operating systems' in BAYER, R., GRAHAM, R.M., and SEEGMULLER, G. (Eds.): *Operating systems. An advanced course* (Springer, New York, 1978), pp. 393-481

ICL Worldwide

The head office of International Computers Limited is at
ICL House, Putney, London SW15 1SW, England

ICL is represented in most countries of the world by subsidiary companies, branches, dealers or agents. The principal addresses are given below.

ARGENTINA	ICL Dealer Sistemas De Informacion, SA, Cerrito 844, Casilla de Correo No. 4305, 1010 Buenos Aires
AUSTRALIA	International Computers (Australia) Pty. Limited 100 Arthur Street, North Sydney, New South Wales 2060, <i>P.O. Box 300, North Sydney, New South Wales 2060</i> (also in Brisbane, Canberra, Melbourne, Perth and Tasmania)
AUSTRIA	ICL International Computers GmbH Meidlinger Hauptstrasse 51-53, 1120 Vienna (also in Feldkirch, Graz, Linz and Salzburg)
BANGLADESH	ICL Registered Agent K Ahmed, 195 Motijheel, Dacca 2, <i>PO Box 309, Dacca 2</i>
BELGIUM	ICL Belgium S.A. Avenue Lloyd George 7, B1050 Brussels (also in Antwerp, Ghent and Liege)
BRAZIL	ICL Do Brasil Internacional Computadores Limitada Rua Washington Luiz, 24-S/Loja, ZC 86, 20.000, Rio de Janeiro, <i>PO Box 3901/20-00, Rio de Janeiro</i> (also in Sao Paulo)
BULGARIA	Branch of International Computers Limited <i>Enquiries to: Bridge House North, Putney Bridge, London SW6 3JX</i>
BURMA	Branch of International Computers Limited 58A Windermere Road, Rangoon, <i>PO Box 542, Rangoon</i>
CANADA	ICL Computers Canada Limited ICL House, 1 Tippet Road, Downsview, Toronto, Ontario M3H 5T2, (also in Calgary, Montreal, Regina and Vancouver)
CARIBBEAN	Branch of International Computers Limited ICL House, 46 Park Street, Port of Spain, Trinidad, <i>PO Box 195, Ports of Spain, Trinidad</i> <i>and</i> 34 Old Hope Road, Kingston 5, Jamaica W1, <i>PO Box 83, Crossroads, Kingston 5, Jamaica W1</i> (also in Barbados, Dominica and St. Vincent)

- CHILE** ICL Dealer
Equipos de Oficina Ltda., Alonso Ovalle 778, OF 21, Casilla 276A,
Correo 21, Santiago
- COLOMBIA** ICL Dealer
Alvaro Fabre G., Friden de Colombia, Ave. Caracas 31-93, Apartado
Aereo 5642, Bogota, D.E.
- CZECHOSLOVAKIA** Branch of International Computers Limited
Ricanova 44, 169 00 Praha 6
- DENMARK** International Computers Limited A/S
Bredgade 23, 1260 Copenhagen K, (also in Horsens)
- EGYPT, ARAB
REPUBLIC OF** Branch of International Computers Limited
1 Abu El Mahasen, El Shazly Street, New Dokki, Cairo, (also in
Alexandria)
- FINLAND** ICL Finland International Computers OY
Annankatu 12A, 00120 Helsinki 12
- FRANCE** ICL (France) International Computers
16 Cours Albert 1er, Paris 75008 (also in Bordeaux, Clermond
Ferrand, Dijon, Lille, Lyons, Marseilles, Nancy, Nates Rennes,
Rouen, Strasbourg, Toulouse and Tours)
- GERMANY, FEDERAL
REPUBLIC OF
(WEST)** ICL Deutschland International Computers GmbH
Marienstrasse 10, Postfach 2466, 8500 Nurnberg, (also offices in
Berlin, Bielefeld, Bremen, Dusseldorf, Frankfurt, Hamburg, Hanover
Mannheim, Munich and Stuttgart)
- GHANA** Branch of International Computers Limited
Mobil House, Accra, *PO Box 117, Accra*
- GREECE** ICL Dealers
Pan Solomos & Co. Ltd., 61 Syngrow Avenue, Athens 404

and
Eurodata Ltd., Singer Building, 63 Stadium Street, Athens 111.
- HOLLAND** International Computers Nederland BV
Zwaansvliet 20, 1081 AP Amsterdam, *PO Box 7113, 1007 JC,
Amsterdam*
- HONG KONG** International Computers Hong Kong Limited
Realty Building, Des Voeux Road Central, Hong Kong, *PO Box 1912,*
- HUNGARY** Branch of International Computers Limited
Regiposta Utca 13, VI 17 Budapest
- INDIA** International Computers (India) Private Limited
Magnet House, Narottam Morarjee Marg, Ballard Estate, Bombay
400038, *PO Box 526 Bombay-400001* (also in Arnedabad,
Bangalore, Calcutta, Madras and New Delhi)

INDONESIA	ICL Dealers P.T. Pansystems, Setiabudi Building, Jl. H. Rangkayo, Rasuna Said, Kunigan, Jakarta-Selatan
IRAQ	Branch of International Computers Limited 8th Floor, Habboo Building, Abu Nawas Street, Baghdad, <i>PO Box 640, Baghdad</i>
IRELAND, REPUBLIC OF	Branch of International Computers Limited ICL House, Adeleide Road, Dublin 2 (also in Cork)
ITALY	ICL Italia International Computers SpA Via Spallanzani, 40, 20129 Milano (also offices in Bologna, Milan, and Rome)
JAPAN	ICL Dealer Dodwell & Co. Limited, <i>Central PO Box 297, Tokyo, Japan</i>
KENYA	International Computers (East Africa) Limited 8th Floor, Bruce House, Standard Street, Nairobi, <i>PO Box 30293, Nairobi,</i>
KUWAIT	Branch of International Computers Limited <i>PO Box 115 (Safat)</i>
MALTA	Branch of International Computers Limited 5th Floor, Development House, Floriana
MALAYSIA	International Computers (Malaysia) Sdn. Bhd. 9th Floor, Wisma Damansara, Jalan Semantan, Kuala Lumpur 23-03
MAURITIUS	Branch of International Computers Limited Anglo Mauritius House, 3rd Floor, Intendance Street, Port Louis
MEXICO	ICL S.A.—Owned 49% by International Computers Limited <i>51% by Mexican Shareholders</i> Avenida Presidente Mazaryk 61, Piso 3RO, Mexico 5, D.F., <i>PO Box 5095, Mexico 5, D.F. (also in Monterrey)</i>
NEW ZEALAND	International Computers (New Zealand) Limited Securities House, The Terrace, Wellington, C1, <i>PO Box 394, Wellington (also in Auckland and Christchurch)</i>
NIGERIA	International Computers (Nigeria) Limited 3rd Floor, Wesley House, 21-2 Marina, Lagos, <i>PO Box 2134, Lagos</i>
NORWAY	ICL Norge A/S Oestensjoeveien 39, Oslo 6, <i>PO Box 36, Bryn, Oslo 6 (also in Bergen)</i>
PAKISTAN	Branch of International Computers Limited Grindlays Bank Building, 1.1 Chundrigar Road, Karachi 2, <i>PO Box 5146, Karachi 2</i>
PAPUA NEW GUINEA	Branch of International Computers (Australia) Pty. Limited 8 Champion Parade, Port Moresby

PHILIPPINES	ICL Dealer Floro Enterprises Incorporated, 500 Carlos Palanca, Quiapo, Manila, <i>PO Box 7241, Manila International Airport</i>
POLAND	Branch of International Computers Limited ICL Warsaw, UL Czarnieckiego 66, 01-548 Warsaw
PORTUGAL	ICL Computadores Limitada Av. Dos Estados Unidos Da, America, 57 A/B, Lisborn 5 (also in Oporto)
PUERTO RICO	ICL Dealer Systronics Business Machines Inc., Calle 272, Lote 12, Urb. Industrial Country Club, Carolina, 00630, <i>PO Box 4096, Carolina</i> <i>00630</i>
ROMANIA	Branch of International Computers Limited <i>Enquiries to: Bridge House North, Putney Bridge, London SW6 3JX</i>
SAUDI ARABIA	Branch of International Computers Limited c/o A Rajab & A Silsilah, PO Box 2815, Riyadh
SINGAPORE	ICL Singapore Private Limited 5th Floor Bangkok Bank Building, Cecil Street, Singapore 1, <i>PO</i> <i>Box 484, Maxwell Road, Singapore 1</i>
SOUTH AFRICA	International Computers (South Africa) (Proprietary) Limited ICL House, 5 Sturdee Avenue, Rosebank, Johannesburg 2196, <i>Private Bag 10, Saxonwold 2132</i> , (also in Benoni, Cape Town, Durban, Port Elizabeth and Pretoria)
SOUTH KOREA	ICL Dealer Taiyang Business Machine Co., 60, 1-Ka, Myung-Dong, Chung Ku, Seoul
SOUTH WEST AFRICA	ICL South West Africa (Proprietary) Limited 5th Floor, Nictus Building, Kaiser Street, Windhoek, 9100, <i>PO Box</i> <i>23017, Windhoek, 9100</i>
SPAIN	ICL Espana International Computers S.A. Avenida Generalisimo 61, 6th Floor, Madrid 16 (also in Barcelona, Seville and Valencia)
SRI LANKA	ICL Dealer International Computers (Ceylon) Limited 20 Sir Chittampalam A Gardiner, Mawatha, Colombo 3, <i>PO Box</i> <i>305, Colombo</i>
SUDAN	Branch of International Computers Limited <i>PO Box 2247, Khartoum</i>
SWAZILAND	ICL Swaziland Limited Independance House, West Street, Mbabane, <i>PO Box 469, Mbabane</i>
SWEDEN	ICL Data A.B. S-171 88 Soina, Sweden (also offices in Gottenburg, Malmo and Vasteras)

SWITZERLAND	ICL (Switzerland) International Computers AG Buckhauserstrasse, 26, 8040 Zurich (also offices in Basel, Bern and Geneva)
SYRIA	ICL Dealer Orcent Computer Division, Shurbatley Buildings, Marjeh Square, <i>PO Box 771, Damascus</i>
TAIWAN	ICL Dealer United Business Computers Limited, Pao-Tung Building, 12th Floor. No. 325, Section 4, Chung-Shiao East Road, Taipei
TANZANIA	International Computers (Tanzania) Limited 10th Floor, NIC Investment House, Independence Avenue, Dar-es-Salaam, <i>PO Box 2569, Dar-es-Salaam</i>
THAILAND	ICL Dealer Bangkok Data Centre Co. Limited, 183 Pitsanuloke Road, Nanleung Bangkok
TURKEY	ICL Dealers Biltek Ltd. Sti., Necatibey Cad 9/2, Ankara <i>and</i> Emak Tickaret Ve Sanyi Ltd., Sti., Halaskargazi Cad 34/4, Harbiye, Istanbul
UGANDA	Branch of International Computers (East Africa) Limited Amber House, Kampala Road, Kampala
UNITED ARAB EMIRATES	Branch of International Computers Limited <i>c/o Mr. Y. Najibi, PO Box 2323, Dubai</i>
UNITED STATES OF AMERICA	ICL Inc. Turnpike Plaza, 197 Highway 18, Third Floor, East Brunswick, New Jersey 08816
URUGUAY	ICL Dealer Arnaldo C Castro, S.A., Casilla de Correo 108, Misiones 1460, Montevideo
USSR	Branch of International Computers Limited UL. Vavilova 83, Kv 4/5, Moscow (<i>Telephone: 346003</i>)
VENEZUELA	ICL Dealer Datatec C.A., Apartado 60216, Caracas
YEMEN, PEOPLE'S DEMOCRATIC REPUBLIC OF (SOUTH)	ICL Representative <i>c/o Aden Refinery Company, PO Box 3003, Aden</i>
YUGOSLAVIA	Branch of International Computers Limited ICL/Mladost, Ilica 28-30, 41000 Zagreb
ZAMBIA	International Computers (Zambia) Limited Provident House, Cairo Road, Lusaka, <i>PO Box 2124, Lusaka</i>

Notes for authors

1 Content

The *ICL Technical Journal* publishes papers of a high technical standard intended for those with a keen interest in and a good working knowledge of computers and computing, but who nevertheless may not be informed on the aspect covered by a given paper.

The content will have some relevance to ICL's business and will be aimed at the technical community and ICL's users and customers. It follows that to be acceptable, papers on more specialised aspects of designs or applications must include some suitable introductory material or references.

The Journal will usually not reprint papers already published, though this does not necessarily exclude papers presented at conferences. It is not necessary for the material to be completely new or original (but see 10, 12 and 13 below). Papers will not reveal matter related to unannounced ICL Products.

2 Authors

Anyone may submit a paper whether employed by ICL or not. The Editor will judge papers on their merits irrespective of origin.

3 Length

Full papers may be of up to 10 000 words, but shorter papers are likely to be more readily accepted. Letters to the Editor and reviews may also be published.

4 Typescript

Papers submitted should be typed in double spacing on one side of A4 paper with full left-hand margin. Mathematical expressions are best written in by hand. Care should be taken to form Greek letters or other unusual symbols clearly. Equations referred to in the text should be numbered. Detailed mathematical treatments should be placed in an Appendix, the results being referred to in the text.

At least two copies should be submitted, both carrying the author's name, title and date of submission.

5 Diagrams and tables

Line diagrams supplied will if necessary be redrawn before publication. Be especially careful to label both axes of any graphs, and mark off the axes with values of the variables where relevant.

All diagrams should be numbered and supplied with a caption. The captions should be typed on a separate sheet forming part of the manuscript. Since diagrams may have to be separated from their manuscript every diagram should have its number, author's name and brief title on the back.

All diagrams and Tables should be referred to in and explained by the text. Tables as well as diagrams should be numbered and appear in the typed MS at the approximate place, at which they are intended to be printed. Captions for Tables are optional. Be careful to ensure the headings of all columns in Tables are clearly labelled and that the units are quoted explicitly in all cases.

6 Abstract

All papers should have an abstract of not more than 200 words. This ought to be suitable for the various abstracting journals to use without alterations.

7 Submission

Before submission authors are strongly urged to have their MSS proof read carefully by a colleague, to detect minor errors or omissions; experience shows that these can be very hard for an author to detect. Two copies of the MS should be sent to the Editor.

8 Referees

The Editor may refer papers to independent referees for comment. If the referee recommends revisions to the draft, the author will be called upon to make those revisions. Minor editorial corrections, e.g. to conform to a house style of spelling or notation, will be made by the Editor. Referees are anonymous.

9 Proofs

Authors will receive printed proofs for correction before publication date.

10 References

Prior work on the subject of any paper should be acknowledged, quoting selected early references. It is an author's responsibility to ensure references are quoted; it will be unusual for a paper to be complete without any references at all.

11 Style

Papers are often seen written in poor or obscure English. The following guidelines may be of help in avoiding the commoner difficulties.

- Be brief.
- Short sentences are better than long ones but on the other hand do not write telegrams.
- Avoid nested relative clauses; preferably start new sentences.
- Define the meaning of ordinary words used in special senses. Define acronyms or sets of initials by quoting the full meaning the first time the initials are mentioned.
- Include a glossary of terms if necessary.
- Avoid words in brackets as much as possible.
- Avoid the frequent use of the type of construction known as a 'buzzword'. This often takes the form of a noun followed by a present or past participle followed by another noun e.g. 'system controlling parameters'.
- Take care in using the word 'it' that the reader will easily understand what 'it' refers to. An unambiguous rule, that cannot always be applied, is that 'it' refers to the nearest preceding noun in the singular.
- Several 'its' in one sentence each used in a different sense can cause considerable confusion. Similar remarks apply to 'this', 'that' and other prepositions.

12 Copyright

Copyright in papers published by the ICL Technical Journal rests with ICL unless specifically agreed otherwise before publication. Publications may be reproduced with permission and with due acknowledgement.

13 Acknowledgements

It is customary to acknowledge the help or advice of others at the end of papers when this is appropriate. If the work described is not that of the author alone it will usually be appropriate to mention this also.

