ICL Technical Journal

Volume 3 Issue 2 November 1982

ICL Technical Journal

Contents

Volume 3 Issue 2

The advance of Information Technology
J.M.M. Pinkerton 119

Computing for the needs of development in the smallholder sector
J.M. Tottle 137

The PERQ workstation and the distributed computing environment
J.M. Loveluck 155

Some techniques for handling encipherment keys
R.W. Jones 175

The use of COBOL for scientific data processing
Tim Harle and Richard Carpenter 189

Recognition of hand-written characters using the DAP
Josef Pecht and Isa Ramm 199

Hardware design faults: A classification and some measurements
T.L. Faulkner, C.W. Bartlett and M. Small 218

ICL Technical Journal

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

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

1982 subscription rates: annual subscription £11.50 UK, £13.50 ($32.00) overseas, airmail supplement £5.00, single copy price £6.75. Cheques should be made out to 'Peter Peregrinus Ltd.', and sent to Peter Peregrinus Ltd., Station House, Nightingale Road, Hitchin, Herts. SG5 1SA, England, Telephone: Hitchin 53331 (s.t.d. 0462 53331).

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

Printed by A.McLay & Co. Ltd., London and Cardiff ISSN 0142–1557

The advance of Information Technology

J.M.M. Pinkerton
ICL Product Marketing Division, Slough, Berkshire

Abstract

The paper is about advances in information technology and some of their likely effects on information systems and on society. Continuing advances in semiconductor logic, in the techniques of designing, fabricating and interconnecting ever smaller and faster microcircuits, in magnetic recording and in digital techniques of communication are driving forces. Concepts such as parallel computing and searching, of non-procedural programming and of 'expert' systems offering advice in difficult areas such as medical diagnosis and drawing on wide collective experience look like becoming practicable over the next decade. There will be improved methods of capturing and presenting digital data. These advances in input/output techniques will not fully keep pace with advances in processing speeds or in the capacity to store data. If equipment goes wrong, it will have been so designed that the equipment itself will indicate this and also which of the parts needs to be replaced. Increases in computer processing speed will be heavily consumed by adoption of methods of commanding machines, that can be used by anyone and everyone. Most literate people, especially in business and administration, will become programmers, but their 'programs' will be expressed implicitly by commands often interspersed in data which they wish to insert, manipulate or obtain. At any given date there are great volumes of intellectual concepts and a colossal range of human knowledge and experience that are not fully put to use. The reason in general is not the lack of an imaginable method of using the knowledge, but rather that no one had yet shown how to assemble that knowledge in a cheap, practical and convenient way. The commercial and social consequences are far harder to foresee than more direct impacts of technological advances. Most of these consequences are however outside the scope of the paper.

1 Introduction

Computer technologists are inclined to see technology – and they usually just mean hardware technology – as the driving force in securing advances in the human condition. For instance, wheels and other mechanisms enabled men to move materials and build pyramids and aqueducts. Printing allowed them to reproduce text and to communicate ideas, and so by improved communication to enlarge the power to govern nations and empires. Electricity first enabled men to signal messages and later to transmit power over considerable distances. I am still unsure whether technology alone is enough to explain such progress. Ideas often come, perhaps must come, before the technical possibility exists for their successful

realisation. Certainly this was true of computers as Charles Babbage's experiments in the nineteenth century showed. His ideas[1] were ahead of the technology of the day. Although they forced the pace and produced great improvements in mechanical engineering, it was not the royal road to success in computing. In our own time Arthur Clarke, the science fiction writer, foresaw the use of geostationary satellites for communication in a celebrated article in *Wireless World* in 1945, two decades before a communications satellite was launched.[2]

Again, although ideas and technology interact so intimately, there are obviously other important factors both limiting and stimulating the application of technical and conceptual advances to practical ends. These include the development of the human skills and experience necessary to use the invention and their progressive formalisation and communication to larger groups of people and ultimately to the whole population. There are obviously economic and commercial factors as well. Social and political factors become important in the later stages, just because so many people's lives can then be affected.

This review tries to paint a picture of what might happen, starting in Part I with computer technology itself. Part II deals with effects on users and usage and on society. In the past technological forecasts for computers, at least for the medium term, have been quite successful in predicting what would happen, and have even been reasonably good on dates. Predictions of commercial viability, or of the popular appeal of inventions, have often been poor or downright bad. So readers should use their own judgment as to what may be the commercial and social factors controlling the rate of application of technical advances.

PART I

2 Technological convergence

What was the technical advance that drove computer engineering and telecommunication engineering into each others arms, so to speak? Fundamentally, I believe this to be a common appreciation of the benefits of digital representation of information of all sorts. In the early days of electronic computing, analogue and digital systems competed with one another. It was not long before digital systems gained total victory because of two factors. First, using enough digits to represent a number any desired degree of accuracy could be attained, and, secondly, using the binary system, regular sequences of pulses or gaps representing say a number can be transmitted and regenerated indefinitely without corruption.

The idea of representing characters by groups of 'noughts' and 'ones' is in fact quite old and goes back to at least the eighteenth century. Its value for communication was recognised by Samuel Morse (although he used dots and dashes). The idea of pulse patterns as a means of encoding the analogue waveforms representing speech or music without error occurred to Alex Reeves in 1938.[3] However, it was not until transistors were invented and manufactured cheaply on a large scale in the 1960s that transmission of speech using pulse-code modulation became a commercial possibility. Techniques for recording and regenerating patterns of pulses

on magnetic media, for instance tape, were invented at the start of computer engineering in the early 1950s and advanced very rapidly thereafter. A computer tape transport today is a very sophisticated piece of equipment.

Not only can a pulse train be amplified and regenerated electronically over and over again without corruption, but extra pulses can easily be added to each group of message pulses, to allow a check for possible errors and allow such errors to be corrected automatically if they occur. So there has been a continuing interplay of innovative advances between computer and communication engineering, that has gradually become more intimate.

Improvements in micro-electronic engineering are only one, though perhaps the most important of these advances. The most important figures regarding silicon devices are:

- speed of response of any circuit, and
- how many circuits can be formed without flaw on one silicon chip.

Factors depending on these are production yield, that m turn determines cost and reliability. In fact manufacturing costs *per circuit* have gone steadily down for at least a decade and show little sign of stopping for the next five or perhaps ten years. To a good first approximation, however, the price of a complete chip has remained constant, but more and more circuits can be placed on each, thus lowering the cost per circuit. In the case of computers, storage elements per chip have gone from one or two to 64,000, and this is certainly not the ultimate limit.

The economic consequences have been more complex. Obviously, it will cost more to design and to set up a manufacturing process for more complex chips. But the complex ones may be needed in smaller numbers. So the custom-built chip becomes expensive and pressures to use standard chips for every purpose become greater. The cost for starting up an advanced semiconductor production facility today can run to hundreds of millions of dollars. With the introduction of more sophisticated techniques this premium increases year by year.

Next, I want to look at improvements in techniques of signalling, that is of communicating data from point to point. Surprisingly, similar considerations apply whether we are talking about passing data from one circuit to another on one silicon chip, or from the earth to the moon, say over a radio link. The delay in transmission is fixed by the velocity of the electromagnetic radiation, i.e. light or radio waves. Faster transmission is not possible, although in some media, e.g. glass, it can of course be slower. For the engineer the important question is how many bits per second can be sent or in older fashioned terms what is the band-width of the channel. Intelligible speech needs a band of about three to four thousand cycles per second (hertz), good quality music say fifteen thousand hertz and a colour TV channel say four to six megahertz. Since modern optic fibres and microwave links can carry probably one to two hundred megahertz at least, methods of sharing a single wide-band link between many conversations or several television signals can be advantageous and have been in use for many years. Equivalent techniques for sharing channels designed to carry streams of pulses representing bits are easily

engineered, for instance where every twenty-fourth bit say corresponds to one telephone conversation or to one data channel. Use of this so-called 'time-division multiplexing' can facilitate the economic design of telephone exchanges if the speech is digitally coded.

It turns out that one cycle per second of band-width can easily carry one bit per second, or with more care and ingenuity two or even four or more bits per second. Thus the faster we want to send data the more band-width it needs and, other things being equal, the more expensive the channel becomes. So the great scope for advance is in getting more band-width cheaply, that is more bits per second for the same money. Communication engineers have studied ways of achieving this intensively for several decades; as a result band-width is getting cheaper to buy in some forms. If we buy it over simple plastic insulated cables, like telephone cables, we can get one or two megahertz of band-width, provided the distances are short, say a few hundred metres. With coaxial cable the band-width can go to ten or a hundred megahertz and over an optic fibre or a microwave link to a few hundred megahertz.[4] But there is always some loss and distortion with increasing distance, so repeaters have to be inserted at intervals varying from a few hundred metres to several kilometres. Microwave range is essentially limited by line of sight, but by putting the repeaters in high orbiting satellites, it becomes world wide. Usually speaking such very wide band-widths are only needed when many conversations or exchanges of data have to be transmitted simultaneously. Whether it is economic for any single user to install his own optic fibre or microwave link, depends therefore very much on the nature and amount of his traffic. For one individual working with a computer a few thousand bits per second is all he can handle, and that only spasmodically. However, if we want to send him very detailed, and especially animated, pictures a band-width of a few megahertz may be needed. When looking at a television picture, for instance, the mind concentrates on a limited portion of the image at any moment. Information displayed on other areas is therefore simply rejected. To transmit large files between centres can also demand wide band-widths. For instance a complete twenty-four hundred foot reel of 6250 bpi tape would take two hundred hours to transmit at 2400 bits/second. Clearly a much wider band is needed than this.

These numbers have important practical consequences. Since most users will have access to computers through modest band-widths (the wider ones naturally attract considerably higher PTT charges) they mean that if users wish to refer to massive public libraries of information, those libraries must clearly be centralised. It is difficult or expensive to update the information in such libraries if it is replicated in computer-accessible form in many different places. However, if the user is a designer working on some elaborate engineering design, he will not be able to afford the delay involved in having his drawings continually transmitted to a remote central computer, amended and re-transmitted back to him. He must manipulate the design information using a local computer having a wide-band channel between it and the display on which he inspects the objects under design. That wide-band channel can then be cheap because it runs between cabinets a few metres apart.

3 General effects of microelectronic advances

There is no space to describe how microcircuits are designed and made. It is

enough to appreciate that on the surface of any silicon chip there can be thousands or tens of thousands of separate circuit elements connected together and that the chip itself is then externally connected to assemblages of many other chips, often mounted on plug-in boards, packages or modules. Since storage elements come in regular arrays while logic circuits that form processors have a much less regular structure, we have special kinds of chips for storage and others for processing. It is convenient for engineering reasons to distinguish these functions, though in nature, in the human brain for instance, storage (i.e. memory) and processing, appear to be more closely integrated together than in computers.

Now microcircuits are sometimes called integrated circuits to distinguish them from circuits built from the 'discrete' or individual components that were used in the early days of computing. The degree of integration is measured say by the number of gating elements formed on one chip. Computer makers originally bought in all or most of the discrete components they needed. Now there has grown up an enormous industry throughout the world supplying integrated circuits. But processor chips are likely to be much more varied and individual than those used for storage. There may be little disadvantage therefore in buying in store chips, because other companies can use similar or identical ones. However, a designer will often want to specify an individual chip for the processing logic, but still avoid the expense of a wholly new chip design. One interesting scheme being developed by many computer makers is called 'uncommitted logic'. This is an array of gates so laid out on the chip that many particular logical arrangements can be impressed on it as a wiring pattern at a late stage of manufacture. Thus one basic design can be put to a wide variety of uses with considerable economy in manufacturing.

The amount of storage required by larger computers is such that several hundred store chips may be needed whereas the similarly complex processor chips required might be only a few. In a small machine a few chips, possibly only one, would form the processor; these chips would also provide a certain amount of fixed storage for microprograms, temporary registers, and so on.

But notice that the interconnections are what actually represent the diversity, i.e. are the essence of a design. The gates and storage elements are all formed to a common pattern. On a chip the interconnections are microscopic conducting tracks that are very difficult to see even in a microphotograph because they are so fine. But many connections between the chips must be made as well and these are still visible to the naked eye.

Just as in the human nervous system where some nerves run one or two metres from the brain (down to the big toe say), we find with computers extensions of the 'on-chip' connections running off the chip altogether, through printed wiring and through cables, between cabinets, in some cases right out of the computer cabinets and via BT cables to other buildings or cities many miles away.

There are certain general effects of the progressive integration of more and more circuits onto one silicon chip. The chance of a chip failing is nearly independent of how many circuits there are on it. It is not absolutely obvious why this is so, but it is true. Also the total number of interconnections *between* chips goes down as the

logic is condensed onto fewer chips; thus the total number of failures in interconnections in plugs and in soldered joints tends to go down as well. At the same time direct manufacturing costs go down, but because the cost of components is not everything, or even a high proportion of the cost a user must pay nowadays, manufacturing costs do not shrink to zero. Because, as was said above, more store chips are needed anyway, the economies of scale apply much more strongly to stores than to processors. To make chip *design* costs negligible in the price charged someone needs to make one hundred thousand or a million identical chips.

The effect of all this on the cost and hence the price of products built cheaply from cheap integrated-circuit chips is therefore rather complex. It is necessary to consider the individual costs involved in producing computers. There will naturally be both direct costs proportional to the numbers produced and a wide variety of fixed costs to be recovered more or less independent of the quantity produced. The reduction in numbers of chips does not affect some of these costs directly, and some are reduced only to a small extent.

So the overall effect on equipment prices will depend very heavily on the numbers to be made, and on the extent to which the designer was able to use standard LSI chips or ones that could be economically customised. Obviously these conditions are much more easily satisfied for small products made on a large scale, like small computers and products for the office or for the home, than they are for larger computers.

At the same time these smaller products will naturally be more reliable, but can afford to incorporate much more elaboration, for example internal fault diagnostics. There will be opportunities to build in more functions and facilities too, even where these are not expected to be intensively used, provided the equipment is made in large enough quantities for design costs to be small compared with production costs.

4 Broad effects of technological convergence

A number of significant changes seem to be coming about because of convergence on common digital techniques by the computing and telecommunication industries, although their eventual effects are hard to assess. It is not just that the use of common methods of design and fabrication in the factory is resulting in mergers or in telecommunication firms going into computers and vice versa. These things are happening of course. More fundamental is the realisation that since all forms of information, speech, music, images, numbers and alphabetical text can be transmitted and processed in common ways, it may sometimes be very valuable to do just that. In the case of television, either speech or vision alone conveys very much less than half of the significance of the programme. Again, although it adds only marginally to the theoretical information content, the additon of colour adds a great deal more to significance.

So what we can expect, especially for the office, are products that enable people to communicate in several ways at the same time, both with one another and with information stored in computer systems.

Since computer engineers and communication engineers are now talking to one another and at last beginning to understand their respective technological jargons, the traditional boundaries between computers and telecommunication equipment, telephone exchanges for instance, are fast becoming hazy. Line communication facilities are being integrated into computer terminals and mainframes. The role of the telephone exchange is no longer merely to switch telephone conversations. It will also be asked to switch computers to other computers, to terminals and to computer peripherals of all sorts.

In parallel with the purely technical changes, governments in many countries are being urged to liberalise their laws and practices regulating the provision of basic telecommunication facilities and for the connection of equipment to public national networks. The lead here has been taken by the United States, closely followed by the U.K. This liberalisation is itself going to encourage the launching of new kinds of service exploiting existing equipment in new ways, as well as stimulating development of quite new classes of equipment.

5 Specific advances in processors and stores

From what has been said it will be clear that to make computers faster, fast enough in fact to be able to keep up with the increasing demands on processing speeds that will be made, they will also have to get physically smaller. Otherwise they will spend too much of the time signalling the occurrence of events in one part of the processor to another part and overall speed will not go up. Today the fastest machines must have a central processor not much over 25 litres (1 cubic foot) in volume. In twenty years time this may be reduced to a few cubic centimetres, and the speed will have gone up in linear proportion to the reduction in the edge of the cube. Amongst other things this will mean a drastic reduction in the power dissipated per switching action, and in the width of the conducting tracks on silicon (or whatever substrate the switching circuits are then formed on). Already these tracks are approaching the smallest size one can form optically, in other words a few wavelengths of visible light. Later on X-rays or beams of electrons rather than light will be used to define these tracks.

However designers will no longer need to limit the amount of logic incorporated so strictly. By adding more they will allow users themselves to diagnose faults automatically. Functional redundancy will ensure that faults do not necessarily cause errors or cause the system to stop dead. There will more often be multiple processors, that is several processors working together so that work can go on even if one or more of them fails. In other cases we shall see arrays of processing elements, as in ICL's Distributed Array Processor (DAP)[5], working in parallel with a common program on the different elements of a single array of values.

Because storage costs are falling rather more rapidly than those of processing, we shall see the amount of storage per unit of processing power going up. Rates of access to data will rise because there will be access to it over more parallel paths than before.

A feature of animal and human brains is the close integration of storage with

processing functions. Cheaper electronics will allow us to do likewise in computers with good effect, as in the DAP for instance. With it one can do sums on whole arrays of numbers, say 4,000 or 16,000 or more of them at one time. This enormous parallelism more than cancels out the reduction in speed of the individual processors and allows the performance of the machine to rival that of the largest computers, or as they are sometimes called 'number crunchers', at a fraction of the cost.

There is another area where doing a number of things at once can readily be imagined and is easily engineered. That is when one is searching for something to fit certain criteria. If one has to reject large numbers of items because they do not fit the criteria, this can quite easily be done on a parallel basis without introducing logical complications. One then has the product ICL calls CAFS, or Content Addressable File Store.[6] It is a kind of data sieve or riddle. After pre-setting what is wanted from a collection of filed data items it will examine and reject items of data flowing past on twelve or more identical parallel paths or channels. Rather than looking for a stored item at a particular place in the store, CAFS looks through a whole section of the store to find a 'key' such as a person's name associated with the wanted information. Any item in a record can be used as a key, such as a person's ability to speak a foreign language. One can search in a single pass for records having unique combinations of characteristics, such as a red-headed Spanish speaker over the age of 50 and living in Manchester or Liverpool. Of course once CAFS has said 'Eureka' and the wanted item (or items) are found, then it is back to serial processing from then on. In many data processing tasks however CAFS seems to offer considerable benefits in the early stages of information retrieval. It is expected to have an important future in these areas of application. It is found that it considerably simplifies the job of programming such searches too.

Summing up then, large and very large scale integration (LSI and VLSI) will bring:

- Usefully more and faster processing per pound
- More parallelism in operations
- More opportunities to provide higher or less common functions such as searches directly by hardware
- Greater reliability and better maintenance facilities, including remote maintenance and automatic repair.

6 Advances in peripherals

Computers both create and function in private worlds – rather as people think their thoughts privately, expressing (or concealing) the results by deeds or words or by gestures and facial expressions.

Similarly computers need some machinery or transducers to receive stimuli from the human world and to produce some visible or tangible expression of the results of processing. When these devices are at hand and employ a data-bearing medium such as paper or magnetic tape, we usually call them peripherals. When they are elsewhere linked by communication circuits and offer visible displays of characters or other images, we commonly call them terminals. Of course, some terminals

include mechanisms like teleprinters. In addition bulk storage devices using reels of tape or rigid and flexible discs coated with magnetic material, are classed as peripherals.

All have been the subject of rapid rates of improvement in reliability, cost and compactness over the past 20 years. It is also unlikely that this rate of enhancement will stop, although there have been and will be considerable differences in rates. For example, mechanical printers printing a line at a time are not going to get very much faster, though the performance of character-at-a-time printers has gone up by five times or more over a decade. Mechanisms have got simpler and less prone to failure; progress has been greatest where the amount of mechanism needed has been least but the influence of electronics and physics greatest. No peripheral mechanism however has improved in price/performance as fast as have electronic processors or stores. Nor is there much prospect of that gap being bridged, though magnetic discs will go on being used and will offer steadily improved compactness and economy for at least another five if not ten years from now.

Other forms of storage are emerging to supplement magnetic methods, including a variety of optical methods. Video discs based on optical techniques have been introduced for home entertainment and offer scope for mass production of replicas. Some of these techniques have been adapted experimentally for use as computing backing stores. Although recordings cannot be erased in some, this may in fact be valuable for archival material, where the objective is to hold records without deterioration for decades or even centuries. Explicit predictions of the nature of future mass storage and archival storage techniques are however difficult.

At the terminals we can expect to see improved visual displays with flat screens and very much enhanced detail, as well as colour where it is needed. There are already on the market screens that will show as much detail as can be typed on an A4 sheet. Soon they will show all the details that one could draw on such a sheet with a fine pencil and offer colour and the ability to move, expand and contract images on different parts of the screen, again in ways we are beginning to see illustrated on the television screen at home. As was pointed out above, this requires both high processing speed adjacent to the screen and a wide-band channel for information between the screen and the computer store. It is interesting that a very high fraction of the brain is reserved for processing visual information. To generate in a computer the large amount of data we can scan by eye each second demands very high processing and display capabilities in the terminal computer. So one expects to see considerable enhancement in both these areas over the next decade.

PART II

7 Introduction

Part II of this paper suggests ways in which expected technological advances may help people to make better use of all sorts of information, not only scientific and technical data but also commercial and administrative data held by innumerable organisations everywhere, and how this may make applying and using computers much easier for ordinary people.

There are three essential functions in all digital computers and digital telecommunication equipment. They are:

(i) Processing, brought about by networks of gates that open or close depending on the combinations of conditions applied
(ii) Storage, that is registering or remembering the result of some previous processing operation, and
(iii) Communication, that is the transfer of results of logic operations performed by the sets of gates and registered in banks of storage elements.

To build the private world of the computer nothing beyond these three basic kinds of function is other than incidental. That is what I sometimes call the doctrine of the Trinity. The cabinets, the cooling and the power supply are life support systems so to speak. One of the three functions dominates a given section of the machine design; thus we recognise the CENTRAL PROCESSOR UNIT, THE STORE, and the INPUT/OUT channels for talking to the human world.

Moreover it seems that there are three basically distinguishable classes of use to which computers can be put. The first is *computing* itself, which is not simply doing arithmetic but more generally the manipulation or alteration of data step by step in accordance with the logic of some program. Data can represent words, numbers, speech, music, pictures and drawings or indeed anything at all that can appeal to the human mind via any of our senses, except smell. All of these can be grist to the processing mill.

The second class of use is *information retrieval.* Notice the difference between data and information; data consist of unrelated items, which in themselves mean nothing. Only when the items stand in some useful and known relation to one another can data represent information. So it is not surprising that computerised information retrieval is bedevilled by problems of establishing inter-relationships between items of data, both before and after they are put away in a computer store.

The third class of use for computers is *communication* as such. Of course communication between people and computers is involved in processing and in information storage and retrieval, but sometimes the aim is direct communication between people, one person and another, one person and a group or between groups; in these cases too computers can be useful. What follows focuses on the prospective benefits to the second and third categories of use, not because processing is unimportant, but because space is limited, the expected advances in processing have been widely debated elsewhere, and because the impacts on society of making information more easily accessible may in the end be more profound.

8 The problems of information retrieval

The problems of computerised information retrieval have their parallels in everyday office existence. They start with that of capturing data; in the office this means making sure you are on the right circulation lists, and that the amount of material entering your office is within your own scope to scan once at least, in the hope that

later you will remember that you have seen it and even perhaps where it was you saw it. For the computer it is a question of putting the information cheaply into machine readable form, if possible automatically. If the event recorded was a physical one, some automatic recorder such as a vehicle sensor in the roadway, a flow recorder or a pressure transducer will do the job. But if it is a mental event, a thought or a wish, then some human act is necessary to capture it. With physical recognition of the words and characters representing continuous speech still a long way off, this has usually meant pressing keys. Thus the importance of text processing is not merely that it makes typists and secretaries more efficient, it can also be a cheap method of capturing data for storage or for processing, as a byproduct of some other useful activity.

In the office, the next question tends to be 'where is that paper I got the day before yesterday and the MD is excited about and on which I must act in the next two hours?'. It is always yesterday or the day before. Had it been today you would remember where the paper was and had it been last month you would have acted on and filed it and so would know automatically where to look for it. This extremely common situation actually encapsulates at least three of the remaining information retrieval problems, viz:

(*a*) how to classify and perhaps format information *before filing it,*
(*b*) how to *index it* in the filing system and,
(*c*) how to *retrive* information whatever detail is taken as the prime key to finding it (in the previous example, age, speaking Spanish, having red hair or living in Manchester or Liverpool).

To find it quickly via an index you have to decide in advance the categories of key that might be important when you come later to look for a missing document. Of course it might not be a single document that was required. It might be an extract from all the documents that referred to a particular topic, for instance all overseas competitors who had brought out a new model of a particular product in the last eighteen months. If you didn't index the filed items under the date of release and the manufacturer's name, you would have to search systematically through a large number of file entries rather than referring to any index to find out where the few relevant entries were. Random searching of course would be a much longer-winded process, sometimes to the extent of making the search too slow to be bearable.

Fortunately there are technical developments to help with all the problems of referring to data items through an index and of searching for it; in practice one often resorts to a combination of indexing and searching which is in effect what one does in looking up a friend's address in the phone book. Knowing the alphabet and the spelling of his name, you use that knowledge to open the right volume quite near the right page and progress rapidly to the page in question. Near the right entry you have to search much more carefully, because there may be several R.J. Jones on the same exchange and perhaps in the same street.

9 Benefits of technology to information retrieval

From what has been said it will be clear that the difficulty and delay in retrieving

information from any set of files involves

- the aggregate size of the files
- how precisely defined was the nature of the item or the items being searched for, and
- how well organised was the system to answer the kind of enquiry now being put; in other words how appropriately chosen were the index or indexes and how amenable was the information filed to classification for filing and how well was that in fact done.

In all cases a certain amount of brute force, i.e. searching through many items is likely to be necessary. If the enquiry is of an unexpected type proportionately more brute force is needed.

Technology can help in various ways. If the computer is faster it can get through the search more quickly, and if the circuits are cheaper one can afford to do more random searching. If the cost of storage is reduced one can hold proportionately more data to meet a requirement for rarer classes of enquiry, and there may be room for more indexes. If the enquiry is about the treatment for some rare form of disease or unusual poison for example, there may be only a few enquiries per day and some of the filed data may get referred to as little as once or twice in a year, but it must nevertheless remain on-line in case of an urgent need.

But there is another basic approach to shortening the response to enquiries. That is to search over many parts of the stored files at once in parallel. If circuits are cheap enough why not have 20 or more searches going on at once? This is the basis for the design of the ICL CAFS product already explained in Part I.

In recent years information scientists have devised special enquiry languages designed to facilitate expressing an enquiry in terms that appeal to people with no professional knowledge of computers. It is too early to claim that these developments in concepts have had a significant effect on design of equipment hardware. There is every reason to think that they will have and that some of these languages will gradually come into very wide use, eventually even with systems like Viewdata intended for ready use by the whole of the population. A forecast like this, however, is not related to technology, but to research into human behaviour and human thought on which something more needs to be said.

10 Ease of use – programming

Over the past ten or fifteen years, the subject of artificial intelligence has attracted increasing attention and has begun to produce very interesting results suggestive of new ways of exploiting collective human knowledge and experience. It has stimulated studies of users' reactions to the behaviour of computer systems especially when used through terminals. The term 'ergonomics' that was formerly thought to mean optimising the form of physical controls and their environment to suit the needs of a human user, is now extended to cover matching his conceptual and intellectual capabilities and preferences ('cognitive ergonomics'). So suppliers are beginning to make systems (and the products that go to build them) easier and

more attractive to use. Technology is being called on to meet requirements framed as a result of advances in the behavioural sciences. To help users find their way through the manuals now held on-line by the computer itself, so as to cope with the consequences of their own mistakes or data errors or system failures, or even to switch the system on from 'cold', all demand still more software. Obeying more instructions will call for more storage. Following a sort of Parkinson's law, programs and data expand to fill the storage available. So here again technical advances make feasible or economic features or facilities that might previously have been thought to be out of the question.

One difficulty in planning a 'helping hand' for completely new computer users is that people's knowledge of how to use a given computer improves rapidly, so that advice and warnings that were welcome to begin with soon become irritating or worse. Thus the 'help' facility has to be carefully planned and open to being worked at several levels. This proves in practice to be quite difficult.

A perspective advocated by the author is for designers to try to visualise the sort of image or model of the way a system works that must exist in the mind of any user. This 'model' must obviously be derived from reality, but the reality is much more complex than the user's model needs to be. So long as the behaviour of the system as seen at a terminal is consistent with a simple model the user will be happy and confident. Once the system starts to behave in a manner that seems inconsistent with the pattern he has learned to expect, he will be unhappy and lose confidence. He will feel in fact that the system is unfriendly. There are two lessons: first, controls need to be natural and obvious in the way they operate and, secondly, a measure of standardisation is highly desirable. These considerations apply equally to the abstract forms of control function provided by system commands say, and to knobs and switches.

Conformance to any standard can contribute significantly to easier use because a user will know better what to expect from the machine or the service he is using. But strict conformance to all standards in a large set will give disproportionately greater benefits, because of the frequent need to bring together data gathered from widely different sources or build program subroutines from a library into a whole program.

After twenty or more years of international effort, supported strongly by suppliers' organisations, a good many of the basic standards necessary have in fact been agreed. These standards form a self-consistent, structured set, with later standards built on earlier ones. Because of commercial pressures and rapid rates of change in techniques however, the level of conformance to standards is uneven. Criteria for conformance are also not widely agreed, so that interworking between different manufacturers' products is not as good as it should be. On the other hand users are increasingly aware of the potential for information and program transfers and are pressing for stricter conformance. In some cases this will be delayed by serious intrinsic difficulties in defining conformance or in devising tests to check for it. This is particularly true for programming languages, and in general for the more abstract aspects of data. It is remarkably difficult to lay down uniform standards for the way all forms of data should be classified, described and expressed if those

standards have to be applied equally to general texts say in a newspaper or in a novel, and to tabulated and graphical information in a technical journal.

Further progress in both framing and in obeying standards is however certain to occur because it is essential for a full exploitation of the possible benefits of computers. These benefits can only come when information of every different kind is open to being related to that of every other kind in any way the human imagination can see sense in.

A most powerful technique for exploiting collective human knowledge and experience by computer is represented by what are called 'expert systems'. So far only a few have been described in the literature.[7,8] They are programs attached to a database that encapsulates the knowledge of many acknowledged experts in a particular field, such as diagnosis of diseases, or the likely causes of faults in a computer say. The technique has been used for example to improve a computer's chess playing technique. The system itself gradually improves its performance as it is used, provided its users are true experts. It does this both by adding to the data base and also by refining the 'rules' to which the system works. These rules are tested through dialogues with experts who may be able to point out the flaws in the generalisations represented by the rules from their own special knowledge of cases. Thus the system can evolve to a degree where its general performance is as good as that of a group of experts collectively, possibly faster than any of them and certainly better than any inexperienced user. In this way, the experts' knowledge and judgment is indirectly made much more widely available. Such systems have obvious applications to the training of specialists in any field using a conversational approach. Obviously in that case, the rules for interpreting the knowledge in the database would not be changed by a dialogue with the trainee, but the training syllabus and methodology might well be.

It must be said that while expert systems show enormous promise, they involve very large programs and a very great deal of work to create the database and bring them into effective operation. New forms of computer hardware will be needed for their full exploitation. So their use may spread only rather gradually over the next decade.

To extend direct use of computers to wider sections of the community, suppliers are looking at ways of allowing programs to be built or adapted by people with little or no professional knowledge of computers. They will often be professionals in other fields – doctors, managers, lawyers, architects, engineers, accountants and so forth. But they will not be prepared to spend much time learning how to program in the older computer languages like Fortran or Cobol, with their arbitrary seeming restrictions on names and fastidious requirements for a rather unnatural formulation of all messages to or from the machine. The research into artificial intelligence already referred to is beginning to suggest ways of avoiding the need to use computer languages, at least those of well established kinds. It looks as if we can expect to see in future schemes of trial and error programming whereby the user merely describes the results he wants to see achieved and can then monitor, step by step, what the system actually does as it does it. If any one stage goes wrong it is apparent and can be put right at once.

An overall judgment is that so far at least, the productivity of programmers seems not to have increased greatly, despite the many superior aids to software development not available ten or fifteen years ago. The number of programmers has certainly increased and it will be open to many more people in future with lower levels of knowledge and skill to devise satisfactory programs. However productivity as such still seems elusive.

11 Some social effects

In his address to an International Symposium organised by UNESCO in Paris in 1979, M. Valery Giscard d'Estaing[9] reviewed a range of impacts of the use of computers on society. One he referred to was its effect on human creativity. Obsessed with the details of how to make computers more biddable and generally house-trained in future, one can easily forget some of the significant effects they have had already. Undoubtedly men would not have walked on the moon without computers, and the development of robots is wholly dependent on them. We know the dependence of government, the banks and many sorts of businesses on computer-based services.

But what will be the effects of computer-aided learning, computer-aided design and draughting and computer aids to information storage and retrieval? Will not the kinds of advance represented by Bousfield's invention of computed X-ray tomography be repeated in many other fields? Will not, in other words, the computer enable us to release our own imaginations from the arbitrary constraints of the human senses and to 'see' and to 'hear' things outside the normal range of our sense organs by reconstructing images and sounds originally detected only by instruments?

The computer may stimulate human creativity in another way by enabling people to communicate and collaborate effectively on research projects and studies though they may live on different continents and rarely meet. Professor Wilkes has spoken to me enthusiastically about his own experience here. Again cheap processing and cheap telecommunications mean an opportunity to assemble and manipulate data and text easily and quickly oneself, without waiting for help from secretaries or librarians. Should this not improve the creativity as well as the effectiveness of administrators and managers?

It is widely believed that computer applications to design and especially to the linking of computer-aided draughting procedures to computer-based manufacturing are rapidly growing in importance. Few industries that have not integrated computers into both their management and the manufacturing and design processes will remain competitive by the end of this century.

Computers have been widely used for the direct control of telephone exchanges for considerably over a decade. In future when nearly all exchanges will use electronic rather than mechanical switching, every exchange will be computer controlled, because that is the easiest way to program it to offer the variety of services subscribers have come to expect.

However the effects of computers on communication itself between people may be of greater significance, and can arise in three main ways. The first and oldest case is that of communication between a single programmer and his users. Here communication is quite indirect via the program, and of course it is almost entirely one way. A programmer nowadays does not often meet most of his users. You might say that the wits of the programmer are put at the disposal of many. The next case is where one person consults a database incorporating facts and knowledge accumulated from the experience of thousands or millions of others. This is a matter of collective experience being drawn on by one individual. The last case, where the practice is not yet common, is the computer-aided conversation or conference. One may visualise two or more people conferring by telephone, aided by a variety of screen-based pictures, not of one another but of the written data or images they want to look at together. Any of these forms of communication can have socially desirable consequences. For example if medical or social services can find out from computers more rapidly who needs help, where that person is and from where the help is best obtained, may they not react more quickly or effectively?

There are of course voices raised suggesting that some effects of computers on society will be bad. They argue that as mechanical automation put men out of work in factories, so now the automation of offices will put people out of work there too. To the extent that they stay at work, it is said, their work may become more routine and more boring as they slave to keep the hungry machines continuously fed with data. To the extent that the programmers and managers have more interesting and varied jobs, they too may face greater strains because of the greater dependence of the whole organisation on the computer for its smooth running. I believe society will in fact be wise enough not to let these things happen and will foresee and forestall them. We do not need to do as certain extreme sects have done, and say "we will have nothing to do with modern technology, no electricity, no main drainage and we will go on using horses instead of tractors".

Others have suggested computers are a threat to personal privacy. They argue this is because it is now much easier to correlate innocent-looking information about individuals from a wide variety of sources. Those facts taken together with a detailed record of one's comings and goings, personal expenditure and say the postmarking of letters, could, in principle at least, allow very revealing or damaging conclusions to be drawn. In the U.K., there seems to be rather little evidence that this threat is real or that most of the electorate is worried about it. So although HMG has commissioned and published more comprehensive studies on the subject than most other governments (the Younger[10] and Lindop[11] Reports), progress towards actual legislation lags behind that in much of Europe and North America. Computers can be programmed to provide all the safeguards of personal privacy anyone could wish for and can also be made highly secure if necessary. However, the risk then is that data is locked up so closely that even those we want to be able to see it may find it difficult to get at.

The real problem is that the kinds of personal information most of us want to see kept private, rely on someone's discretion. The building of discretion into computer programs is not feasible as far as I can see, if by discretion we mean exercising judgment about revealing some piece of information based on the circumstances

of the case itself. Though there have been privacy guidelines, there have been few published examples of exact rules governing access to private data held by computers. A computer however can only be programmed in terms of rules, not guidelines.[12] Perhaps the debate going on amongst doctors indicates real difficulties in framing satisfactory rules.

12 Summary of effects on computer operation and usage

A very pronounced trend in the 1970s, which is certainly not fully complete, has been a rapid distribution of computing functions widely within organisations and within communities.[13] In a few special types of organisation there is already one terminal per employee. Many offices provide a telephone for every employee today and whilst the unity ratio for terminals will not be attained in most organisations even by the end of the century, it is likely to be reached eventually. Most of those terminals will incorporate a key pad, a microcomputer, some storage and a limited form of display screen and many will incorporate a telephone.

Whilst the majority of users will not themselves write programs in the classical way some of them will address complex queries or sets of commands to the system. These will be interpreted by the system as if they were specially written programs. In the office computers will help people to keep track of their documents, to process them and send them to other offices more easily, quickly and reliably than today. But paper may vanish from offices much more slowly than the more optimistic forecasts suggest.

Computers will be used via cheap data communication links from all over the world; they will talk to users in any alphabet a user prefers, be it Roman, Arabic, Cyrillic, Hebrew, Japanese, Chinese, or whatever. Few nations will be excluded, because there will be adequate, possibly superb, facilities for translation between any of the commoner natural languages.

As machines and terminals become easier to use, more reliable and self-repairing, the pervasive influence of computers will be largely unseen. It will be as much taken for granted as domestic power wiring or plumbing. Central facilities will be just as unconsidered as the control room and its equipment in a television studio, or at an electricity generating station. Only specialists will be interested in the detail of what goes on in these centres. Information and other services supported by computers however will be regarded as another essential public services like the electricity supply or main drainage – and will be allowed to fail equally seldom.

Acknowledgement

The paper is a slightly modified version of the script of an address given to the Institute of Water Pollution Control at their National Symposium, April 1982. The author, and the Editor of the ICL Technical Journal, are most grateful to the Institute, and in particular to J.C.Holding, Hon. Secretary of the East Anglian Branch, for permission to publish it here.

References

1 A full account of Babbage's work is given in BOWDEN, B.V. (ed.); *'Faster than Thought'*, published Pitman, London 1953. p. 7 et seq.
2 CLARKE, Arthur C.: Wireless World Vol 51. Oct. 1945.
3 REEVES, A.H. *et al.:* "Improvements in or related to Electrical Signalling Systems". British Patent 509820 Appn date Nov. 1937; granted July 1939.
4 WHYTE, J.S.: 'Telecommunication – the way ahead'. *Electronics & Power* 1980, **26**, p.719-726.
5 SCARROTT, G.G.: 'Wind of Change'. *ICL Tech J.*, 1978 **1**, 35
6 MALLER, V.A.J.: 'The content addressable file store – CAFS'. *ICL Tech J.*, 1979, **1**, 265
7 MICHIE, D.: 'Expert knowledge re-engineered'. *Comp. Bull.*, 1981, II/28, 6
8 ADDIS, T.R.: 'Towards an expert diagnostic system', *ICL Tech. J.*, 1981, **2**, 79
9 d'ESTAING, V.G.: Speech to UNESCO Symposium on Information Technology and Society, Paris 28.9.1979
10 YOUNGER, K.: 'Report of the Committee on Privacy'. HMSO London, Cmd 5012, 1972
11 LINDOP, N.L.: 'Report of the Committee on Data Protection', HMSO London, Cmd 7341 1978
12 PINKERTON, J.M.M.: 'Security and privacy of data held in computers', *ICL Tech. J.*, 1981, **2**, 3
13 WILKES, M.V.: 'Computers into the 1980s', *Electronics & Power*, **26**, 67-71

Computing for the needs of development in the smallholder sector

G.P. Tottle
ICL Applications Systems Division, Manchester

Abstract

A paper in the May 1979 issue of the *ICL Technical Journal* (Vol. 1 No. 2 Tottle, G.P: 'Computers in support of agriculture in developing countries') described a research project into generalised systems to support smallholder-based agricultural development. The project was undertaken jointly by ICL, the University of Manchester Institute of Science and Technology (UMIST) and individual agriculturalists. The system, SCAPA – System for Computer-aided Agricultural Planning and Action – has now been implemented and has been in successful pilot use for a year with the Rubber Industry Smallholders Development Authority (RISDA) in Malaysia. The Authority is concerned with the general well-being of the smallholders in addition to its specific rubber-production objectives which involve managing a quarter of the world's supply, and is in many respects a leader among world authorities in development.

The present paper describes SCAPA as now implemented and enlarges on the absorbing range of different environments and requirements now involved. In summary, the system covers planning and production for the main farmer-supported functions: extension monitoring, input supply, credit management, marketing and research.

1 Background

SCAPA (System for Computer Aided Agricultural Planning and Action) is a project set up by ICL Research Division, International Division and Manchester University in 1978 to examine what contribution computing could make to agricultural development particularly in the smallholder sector in developing countries. It has involved an interesting and unusual mix of computing and operational research specialists and those from various agricultural backgrounds, ranging from researchers to people with twenty or more years of field experience as agricultural or administrative officers in developing countries. Their proposals have benefited greatly from inputs and criticism from ODA and the World Bank, from development institutes, consultancies and universities in the UK and overseas, and in particular from visits and detailed studies with development authorities in several countries.

These studies gave rise to a conviction that, while the areas of macro-level modelling, organisational planning, economic analysis and forecasting had been well explored, the practical use of computers to meet the informatics requirements of grass-roots development in the field had been relatively neglected. In our field studies therefore we have worked from the farm level requirements upwards. The farmer's view of development is illustrated in Figure 1, which shows the various functional units (primarily planning, input supply, extension advice and marketing) on whose activities he depends.

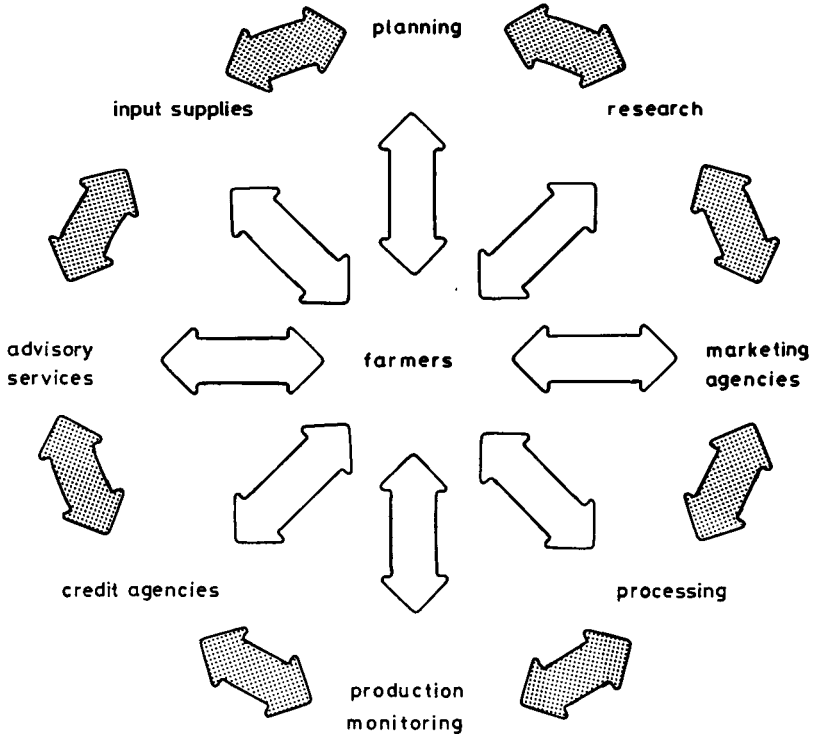


Fig. 1 Information flow in supporting small farm agriculture

The accepted view when the project started was that computers were the tools of affluence, too costly for application in developing countries, too complex, too demanding of specialist skills, too demanding of the support environment, too demanding of literacy and numeracy in the population, too inimical to clerical employment where clerical workers are relatively low-paid and clerical work provides a socially and politically important source of job opportunities. Technology had to be appropriate, and effort should be concentrated on developing indigenous agricultural systems, or on severely practical improvements such as the introduction of high yield varieties of crops or locally available fertilisers and agricultural inputs or of cheap, locally manufactured agricultural equpment.

Despite the immense success of many practical programmes, however, overall poverty levels have increased steadily; to quote the President of the World Bank in 1980[1] 'The absolute poor are not a tiny minority of the massive population in South East Asia, India, Africa and South and Central America; on the contrary, they constitute about forty percent of the two and one quarter billion[2] individuals living in the developing countries: forty percent of the populations of the developing countries who have neither been able to contribute significantly to national income growth nor to share equitably in economic progress.'

Our conclusions from our field and academic studies lined up with the view, now general, that the key problem areas in most rural development concerned *local* management. They concerned the human requirements of advising farmers and identifying and agreeing their intentions, and the logistical problems in servicing these myriad intentions. Central to the improvement of this local management capability is the question of information flow, of providing timely and accurate information across large numbers of people.

2 The Product, SCAPA

SCAPA has been implemented with these requirements foremost, and the remainder of this paper gives summaries of the facilities and a fuller discussion of the potential user situations which we have encountered in the five countries in which our investigations have gone into greatest depth. What one software package can achieve in this field is clearly limited against this daunting backdrop. The user requirements in this area of application are however, in many respects very specialised and this fact has not as yet been matched by appropriate software.

2.1 Physical Communications

SCAPA aims in the best situation to enable small-holders to communicate directly with the computing system, but where this is not feasible, to provide tools for advisory and support staff to do so on the small-holders' behalf, if necessary on a sample basis. This reporting via intermediaries is essential where transport, literacy or costs so dictate, and is necessary in any case for large estate or communal farm applications. The base unit is normally a 'plot' within an 'individual' farm. In many developing societies both concepts are often elusive, but the fact that computing systems can handle diversity at low levels, whereas human clerical systems cannot, is potentially very important. Commonly, smallholders in development projects are forced into agricultural straitjackets – a single crop package offering one or perhaps two high yield seed strains, for a standard area, assuming standard soils, prescribed clearance, planting, harvesting and other dates, with fixed interest rates, loan repayments and so on.

This constraint on diversity is, under manually based management systems, a matter of administrative necessity. And yet it locks out 'the wealth of knowledge and validity of insight embodied in indigenous technical knowledge, which the outside

expert often lacks'.[1] These indigenous technical systems are often highly sophisticated, effective and satisfying – as an example the Mexican Chinampa in Fig. 2.[2] It may prove that information technology enables this constraint on diversity and indigenous technical knowledge to be relaxed; our experience with RISDA (the Rubber Industry Smallholders Development Authority) in Malaysia indicates that this is so as shown later in this paper. The research carried out in Nigeria described by Bede N. Okigbo[3] brings home the major differences between Western maximum yield systems and the tropical requirements, in research, crop management or mechanisation, 'to accept less than the best return from any one part of the system in order to maximise the return on resources for the system as a whole'.*

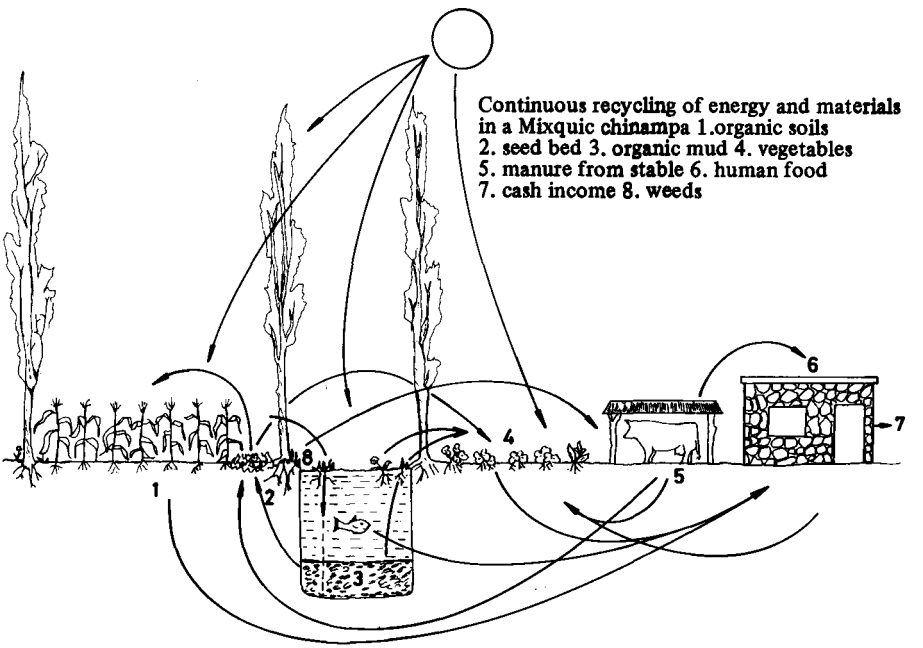


Fig. 2

In the package we have tried to provide facilities which cater for the current mainstream of requirement by development authorities and estates for the closely managed single-crop system, but to leave open the options to cover greater diversity of activity if needed. This leads towards the paths predicted by futurists for 'rural development terminals' at which smallholders can enquire the right agricultural strategies for their own plots, fix on that which attracts most, and then get the logistical support which is commonly the biggest obstacle to achieving this strategy.

Physical communications are a major constraint in all this. Typically many users will aim at most at once-monthly interactions with their smallholders, use bus

*J.D. von Pischke makes a related point on the importance of appropriate advice and of local management from his experience in Kenya, where maize yields were more than double by developing local practices, without injecting external credit, fertiliser or high yield seed varieties.

services or trucks to transport paper work, accept fairly frequent breaks in communications (interaction with even 20% of smallholders would be a major improvement on the current information flow), and work if need be on a stratified sample basis. However, in Malaysia and Kenya more ambitious approaches using micros, e.g. DRS, at district headquarters communicating by land-line to regional mainframes are under development. In general the two major communications obstacles to the approach which we expected, (i.e. to direct farmer-to-system interaction) smallholder literacy and physical transport, have proved surprisingly surmountable.

2.2 *Facilities*

SCAPA comprises a Basic System, which targets on the requirements for planning and monitoring progress, and three optional sub-systems; the Input Supply Subsystem (ISS), Credit Subsystem(CSS), and Marketing Subsystem (MSS). This paper covers all four components, but users can select those they need and the primary initial requirement for most is the Credit Subsystem with a growth path to the ISS in particular.

2.2.1 Database: A prime need recognised by the SCAPA project is to have information appropriate to the resources, planning and productive activities of individual

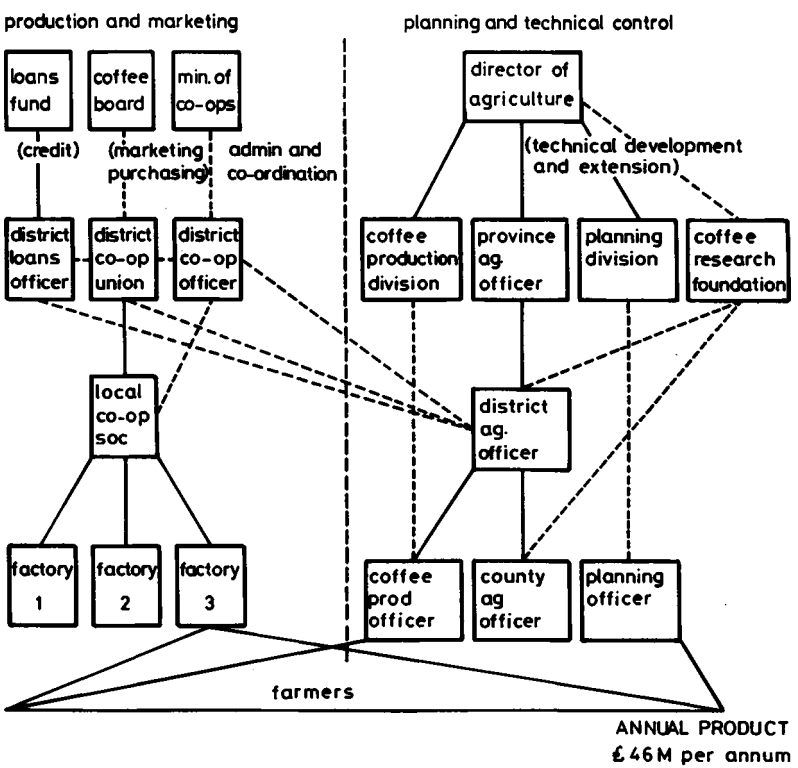


Fig. 3 Organisation of coffee production - Murang'a, Kenya

farms in the form of a common data-base and codings. Frequently, in manual systems, large numbers of files are held on different aspects of farmers' resources and activities without such a common information base for cross reference. Capturing data in sufficient detail is expensive but it provides a basic permanent record against which the continuing stream of technical possibilities can be evaluated for the future.

The three main files are PROFILES, ACTION LISTS and PLANS. All are structured hierarchically in the tree: District, Nucleus, Farm, Plot.* The structuring is a feature absent from other, primarily statistical analysis, systems we have encountered which work at a single level. It is particularly important because it maps well onto the management structure of most users and estates, as illustrated in Fig. 3, and enables abstracts to be produced cheaply giving the important information at the relevant level – for example the District Agricultural officer to get summaries of the total acreage of coffee sprayed with CBD fungicide for his 48,000 farms at a given date. Further, it enables the system to support situations, as in agricultural production in Egypt, where land fragmentation prevents operation at the individual farm level, or Zimbabwe where the base unit for operational purposes is the farming group and where administration and accounting even at this level is a major bottleneck, or as in Uganda where, for the EDF's coffee rehabilitation project, the need is for a high level logistical support system covering national, regional and district levels.

2.2.2 Profiles: The profiles relate to the resources of individual farmers and their farms. The value of acquiring such information for rational planning of farming is frequently recognised, and a number of different sets of proposed information parameters exists (such as that in FAO's FARMAP). In SCAPA we separate the information to be compiled from the farm profile into some mandatory items, which must be obtained and entered during set-up, while other information is optional according to the user's perception both of specific requirements for the project and what it is practically possible to obtain.

The mandatory information includes the farmer's name, address, identification, farm area, tenure type, area under cultivation, total area and the area of each plot subject to a monitored SCAPA plan. Most of this information is often available in existing manual systems and can readily be transferred. Optional farm records allow users the opportunity to record within the profile, a great range of potentially valuable information, including the following: distance from the nearest headquarters or farm service centre; distance to the nearest road; the seasonal passability of the nearest road; the labour availability anticipated on the farm by month; the educational and training characteristics of the farmer; the altitude of the farm; the water supply available; the motive power available. Space is provided within the files for other user-defined optional records. Optional plot records provide for the recording of a full set of soil characteristics, aspect, shade, prevailing wind, previous crop history and spare information of the users choice.

*Any 4-level structures based on other defined strata can be handled by the same software.

GENERAL ACTION LIST GROUNDNUTS NEW 000238 PAGE 15

CASH CONVERSION FACTOR 1

PERIOD NUMBER	PERIOD QUALIFIER	ACTION NUMBER	INPUT FLAGS	OUTPUT FLAGS	RESOURCES ITEM	QUANTITY	MANAGEMENT CODE	REPORTING CODE	RESPONSIBILITY CODE	TEMPLATE ACTION	CRITICAL DATE	NECESSARY PREDECESSORS	CREDIT FACTOR	CREDIT FLAG	MANDAYS
						ACTIVITY									
			CONFIRM RECEIPT OF BENOMYL FUNGICIDE.												
12	A	15	0		80224	0.00		S	S	0	15/12/00	7	8-	L	0
			DELIVERYOF GROUNDNUT SEED.												
12	A	16	C		00000	0.00		F	F	0	15/12/00	15	0		0
			CONFIRM RECEIPT OF GROUNDNUT SEED.												
12	A	17			00000	0.00				0	30/12/00	10, 16	12	L	20
			SOW GROUNDNUT SEEDS 5-10CM DEEP (2-4 INCHES) AND 10 CM APART ALONG THE RIDGES (4 INCHES).												
12	A	18			00000	0.00				0	05/01/01	12, 17	0		20
			APPLY PHOSPHATE FERTILIZER IN A RING 10CM IN DIAMETER (4 INCHES) ROUND EACH GROUNDNUT SEED.												
12	A	19			00000	0.00				0	20/01/01		0		6
			FIRST WEEDING OF GROUNDNUTS. REMOVE ALL DEBRIS FROM THE FIELD.												
12	A	20			00000	0.00				0	30/01/01	13	0		2
			FIRST SPRAY OF BENOMYL FUNGICIDE ON GROUNDNUTS. MIX 226GM IN 180 LITRES OF WATER (0.5 LBIN 40 GALLONS).												
1	B	21			00000	0.00				0	20/02/01		0		2
			2ND SPRAY OF BENOMYL FUNGICIDE ON GROUNDNUTS (14 DAYS AFTER THE 1ST).												
1	B	22			00000	0.00				0	20/02/01		0		8
			2ND WEEDING OF GROUNDNUTS. EARTH UP THE BASE OF THE PLANTS AND REMOVE ALL WEED DEBRIS FROM THE FIELD												
1	B	23			00000	0.00				0	01/03/01		0		2
			3RD SPRAY OF BENOMYL ON GROUNDNUTS (14 DAYS AFTER THE 2ND SPRAY).												
1	B	24			00000	0.00				0	15/03/01		0		5
			LAST WEEDING OF GROUNDNUTS. REMOVE WEED DEBRIS FROM FIELD. EARTH UP BASE OF PLANTS. SO AS NOT TO DISTURB DEVELOPING GROUNDNUTS, DO NOT WEED ANY												
2	B	25			00000	0.00				0	20/03/01		0		2
			4TH SPRAY OF BENOMYL ON GROUNDNUTS (14 DAYS AFTER THE 3RD												
2	B	26			00000	0.00				0	28/02/01		0		0

Fig. 4 General Action List for groundnuts in Malawi

2.2.3 Action lists: The next set of information, which should be based on a combination of research and experience, concerns the detailed specification of the main actions required for growing a particular crop or crop combinations under specified environmental conditions. In SCAPA such listings are described as General Action Lists.

Whereas General Action Lists will detail crop requirements in terms of unit areas, farmers will normally be growing one or more than one crop on irregular or fractional areas and their Farm Action Lists will be drawn up in consultation between the farmers themselves and the project staff based on the requirements and actions for growing particular cropping systems on specified areas of land. Part of a general Action List for groundnuts in Malawi is shown in Fig. 4: a simplified version is given to the farmers, excluding information irrelevant to them.

The preparation of farm action lists requires, in effect, co-operation between the research staff, the extension staff and smallholders themselves. If SCAPA is used as a tool in a base-line study of existing farming systems, it may be that the action list is simply a summary in calendar form of the actions in fact being carried out by farmers without extension intervention. Action lists, where there is a package of practice considered by the research and extension staff to be capable of improving the productivity and economic position of smallholders, will on the other hand list out in calendar form those actions which the smallholder, in discussion with the extension staff, has agreed to undertake. When discussion with the farmer has been completed, the action list printed by the computer (in the farmer's own language) includes for each action the month when the action should be done, the number of the action, an action description, a date by which it should be completed in order to be effective, the action number of necessary predecessors, the approximate number of man-days for the action on the plot in question and a credit factor that may be applied to the action as a means of metering out credit according to the level of achievement of his own plan by the farmer. The printed action list becomes a key document of which copies are held by both the smallholder and the extension staff, and is the basis on which assistance to the farmer is planned and executed.

2.2.4 Farm plans: During the planning phase in a SCAPA-assisted project, farmers select action lists, modified if they wish, to comprise their own farm plans. The computer program then challenges farm action lists individually against farm profiles. Suggested farm action lists that can not in fact be carried out within the available resources, or transgress other bounds such as the plot's soil series, the gradient or the aspect, are flagged and can be revised. A similar process, to plan and to check plan viability, is carried out manually in Kenya for applicants for large agricultural loans; typically a district's skilled five-man Farm Planning team covers about 500 farms a year, giving excellent service but covering at most a tenth of those who need it.

Once an original or revised action list is accepted, it becomes the agreed basis for a farmer's action and responsibility, but is neverthless capable of being modified during the farm year in response to external events such as unexpected timing of the rains, or as a response to actions which have been reported as departing

from those agreed. For example, an agreed action of applying fertilizer might be deleted from an action list if the necessary predecessor for economic application of fertilizers, such as pruning, had not been carried out.

These three main files are complemented by files under the control of specialist staff:

The RESOURCES file (Input Supply Subsystem) containing details of agricultural inputs – name, unit cost, availability, supplier and a chain of alternative products. This completes, on the supplies side, the information necessary to aggregate supply requirements over time; in no situation we have encountered in 15 countries were the current manual systems considered by project staff to work satisfactorily. This was so even in a fairly static situation in which farmers were tied to fixed plots and standard packages. To those familiar with computing, the ease will be obvious with which variations in price, time, transport arrangements and so on can quickly be fed into a computer-based system and the effects evaluated. So will the additional advantages of iterating the process to seek the better and more equitable fit. Nevertheless, under manual systems the volume of data, its unreliability, and its relative inaccessibility prevent the process except at macro levels, and a gap between real needs and management plans results. The cost in lost production is frequently in the region of 30%. This loss occurs for instance in Zimbabwe when a wheat grower receives seed of a high yield variety which is particularly sensitive to planting dates, despite the fact that his own constraints on labour make it impossible to plant within the time window required.

The SALES file (Marketing subsystem) contains details of farm outputs, prices, grades and buyers. This mirrors the functions carried out by the ISS but on the output side. Marketing in developing countries meets similar problems to those on the supply side, compounded by the fact that the function is the last in a hardly predictable chain, and that market characteristics fluctuate relative to additional external factors such as overseas demand and supply levels or government pricing policy. Nevertheless a mechanism is valuable, for example to provide for transport and processing costs to be predicted, recorded and deducted and potential gross margins to be budgeted and monitored. This is rarely done at the farm level in manual systems. The requirement however is increasingly evident as smallholders move from subsistence towards 'emergent commercial' farming.

The CREDIT file (Credit subsystem) contains details of the farmer's loan application, authorisation, credit limit, current indebtedness, interest rate, repayment schedule and repayment history. Again the flexibility introduced by electronic processing rather than clerical systems is immensely important.* Under one relatively successful manual scheme the farmers are tied to a fixed annual interest deduction, whether the loan lasts 12 months or two, whereas for RISDA, interest is computed at a daily rate and varied according to the loan purpose. This enables the

*The way in which the computing requirements for small farmers accounting differ from those for large western-style farms is discussed in Reference 14, where Mr. Keith Evans describes the Zimbabwean Agricultural Finance Corporation's AFIS system in detail.

SCAPA GENERAL ACTION LIST KACANG TANAH 000006

PERIOD NUMBER	ACTION NUMBER	ACTION
2	1	MEMBUAT PERMOHONAN SEBSIDI UNTUK TANAMAN KACANG TANAH SEBAGAI KONTAN.
2	2	MENERIMA KELULUSAN SUBSIDI UNTUK TANAMAN KACANG TANAH SEBAGAI TANAMAN KONTAN.
3	3	MEMOHON BEKALAN BENIH KACANG TANAH, 80 KG SEHEKTAR.
3	4	MEMOHON KAPOR, 1520KG SEHEKTAR.
3	5	MEMOHON BAJA CCM83, 7 BAG SEHEKTAR.
3	6	MEMOHON RACUN SERANGGA BIDRIN, 2.7 LITER SEHEKTAR DARIPADA KEDAI.
3	7	MEMOHON RACUN KULAT BENLATE 1/2KG SEHEKTAR DARIPADA KEDAI.
3	8	MEMOHON RACUN RUMPAI PRA-CAMBAH LASSO, 4.2 LITER SEHEKTAR DARIPADA KEDAI.
4	9	MENERIMA BEKALAN BENIH KACANG TANAH, 80 KG SEHEKTAR.
4	10	MENERIMA BEKALAN KAPOR 1520KG SEHEKTAR.
4	11	MENERIMA RACUN RUMPAI PRA-CAMBAH, 4.2 LITER SEHEKTAR DARIPADA KEDAI.
4	12	MENERIMA BAJA CCM83, 7 BAG SEHEKTAR.
4	13	MENERIMA RACUN SERANGGA BIDRIN DARIPADA KEDAI.
4	14	MENERIMA RACUN KULAT BENLATE, 0.5KG SEHEKTAR DARIPADA KEDAI.
4	15	MEMOHON TRAKTOR UNTUK MENYIKAT TANAH.
5	16	MENYIKAT TANAH PERTAMA.
5	17	MENABOR KAPOR 2 HINGGA 4 MINGGU SABELUM MENANAM, 1520 KG SEHEKTAR.
5	18	MENYIKAT TANAH KEDUA.
5	19	MENYEMBOR RACUN RUMPAI PRA-CAMBAH LASSO, 4.2LITER SEHEKTAR.
5	20	MENABOR BAJA BASAL CCM83, 7 BAG SEHEKTAR.
6	21	MENANAM BENIH KACANG TANAH 1X1 KAKI, 80KG SEHEKTAR.
7	22	MENYEMBOR RACUN SERANGGA BIDRIN.
7	23	MENYEMBUR RACUN KULAT BENLATE, 1/2KG SEHEKTAR.
7	24	MENGAWAL RUMPAI SECARA MANUAL.
8	25	MEMUNGUT HASIL.

Fig. 5

authority to cover 23 different crops and livestock systems and to weight credit in favour of the smaller holdings and poorer farmers, to encourage particularly desirable crops and mixes (rubber interplanted with groundnuts for example) and to discourage others like cassava which were considered to provide high initial returns at the expense of long term soil degradation.

The advantages in security against fraud or error and in speed of turn round, accuracy and cheapness deriving from computer procedures do not need elaboration. The best manual systems involve substantial delays in loan disbursement and recovery; in worst cases funds are pent up behind 3-year backlogs of loan applications awaiting processing, while in others incentives to individual farmers are reduced markedly by the fact that manual administrative systems permit payment only on the basis of large groups, regardless of individual achievement.

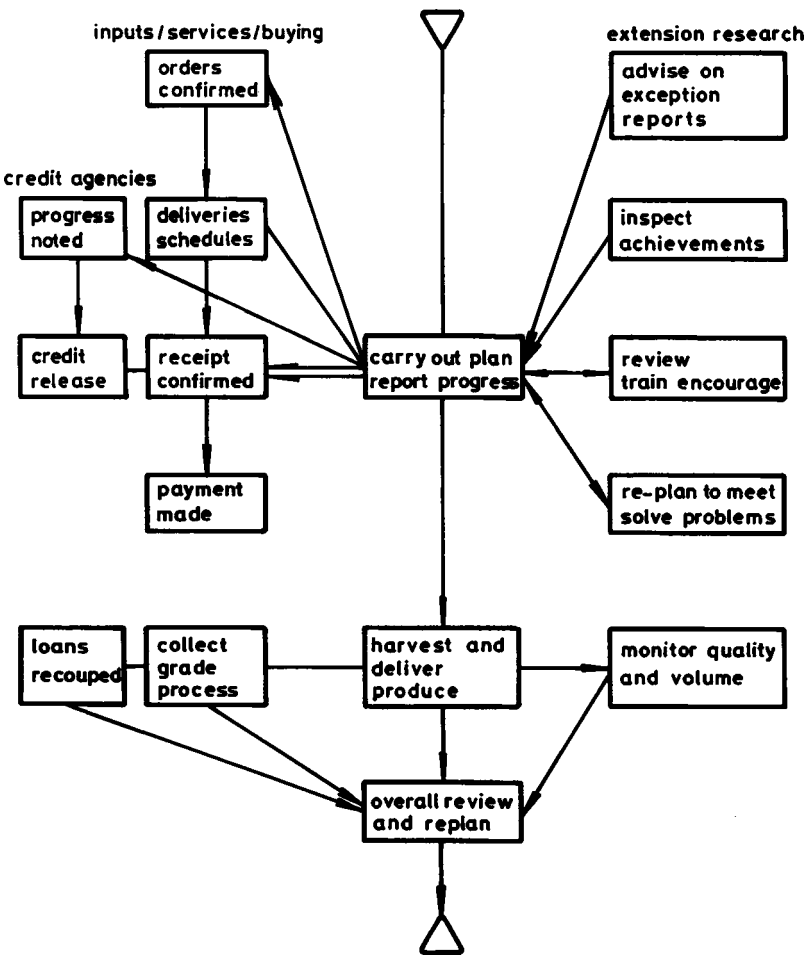


Fig. 6 SCAPA production phase — monthly review cycle

```
                    SCAPA REPORT FOR PERIOD        03  OF     85/08/32

      FARM IDENTIFIER: 034750200477              NUCLEUS: KAMUGANGUZI
    FARMER: MR E.NJORO
   ADDRESS: MILE 12,BUKIMBIRI RD,KISORO
                  PLOT:   2    FOR ACTION LIST:          BURLEY TOBACCO, MALA
ACTIONS COMPLETED LAST PERIOD:
-----------------------------

   NO ACTIONS

ACTIONS NEXT PERIOD (FROM YOUR AGREED PLAN)
-------------------------------------------
NO.    DESCRIPTION                                          CREDIT CRITICAL     NECESSARY
                                                            VALUE  DATE         PREDECESSOR(S)
   5 CUT BRUSHWOOD FOR STERILISING NURSERY BEDS AND STA             31/09/81
     CK NEAR NURSERY SITE.

   6 CUT SHORT GRASS MULCH FOR NURSERY BEDS.                        31/09/81

   7 PREPARE MATERIALS FOR CONSTRUCTING WINDBREAK FOR               10/04/81
     COMMUNAL NURSERY SITE.

   8 CONFIRM ORDER FOR SEED.                                        15/09/81

   9 CONFIRM ORDER FOR CAN FERTILISER (1BAG-50 KG)                  10/09/81    3

  10 CONFIRM ORDER FOR 20:20:0 FERTILISER (1BAG 50 KG)              10/09/81    4

  11 INSPECTION OF COMMUNAL NURSERY AREA BY STA                     20/09/81    8   9  10
     FF,GIVE TRAINING TO FARMERS ON NURSERY OPERATIONS.

  12 DELIVERY OF TOBACCO SEED.                                      30/09/81    2   8

  13 DELIVER CAN FERTILISER.                                        30/09/81    3   9

  14 DELIVER 20:20:0 FERILISER.                                     30/09/81    4  10

  15 PREPARE NURSERY BED 25YDS LONG 1.3 YDS WIDE (25SQM            15/10/81    5
     ) AND LAY DRY GRASS AND BRUSHWOOD FOR BURNING TO 2
     FT DEEP OVER SOIL. STERILISE BED BY BURNING.
```

Fig. 7

SCAPA REVIEW FOR PERIOD 3 ON 05/08/82

FARMER: MR E.NJORO *034750200477 002 03A*

ACTIONS NOT YET COMPLETED:

NO.	DESCRIPTION	CREDIT VALUE	CRITICAL DATE	NECESSARY PREDECESSOR(S)	ACTION NO.	REPORT BOXES Y	N
1	AGREE WITH TOBACCO AUTHORITY TO GROW 1 ACRE OF TOB ACCO (HEAVY WESTERN/FIRE CURED) SELECT PLOT.		31/07/81		*001*	✓	

ACTIONS DUE FOR CURRENT PERIOD

NO.	DESCRIPTION	CREDIT VALUE	CRITICAL DATE	NECESSARY PREDECESSOR(S)	ACTION NO.	REPORT BOXES Y	N
2	ORDER TOBACCO SEED (6 GMS FOR 32.5 SQ YDS (=25SQ M)FOR 1 ACRE PRODUCTION).		15/08/81	1	*002*	✓	
3	ORDER 1BAG(50KG)CAN.		15/08/81	1	*003*	✓	
4	ORDER 1BAG (50KG) 20:20:0 FERTILISER.		15/08/81	1	*004*	✓	

PLEASE NOW ADD ANY INFORMATION ABOUT PROBLEMS WHICH YOU HAVE ENCOUNTERED OR ANY OTHER INFORMATION FOR FIELD STAFF

SIGNED E. NJORO DATE 14/08/81

Fig. 7 Farmer's monthly review sheet

In summary then, the six main components of the SCAPA database – PROFILES, ACTION LISTS, PLAN, RESOURCES, CREDIT and SALES – provide for data covering as many of the major functions as the user wishes.

2.3 Production

The system can be used in a variety of ways and to match a variety of reporting intervals – we are currently checking out a weekly based system covering chicken production for example. The implementation for RISDA in Malaysia and the Resettlement Loans Fund in Zimbabwe are more typical and work on a monthly basis as illustrated in Fig. 6. Each month the computer generates a letter which is sent by bus to the farmers, showing actions completed last month, those due next month, a broadcast of any special messages and a report for the current month. The farmer (or extension agent on his behalf) ticks off the report section indicating those activities he has completed and those outstanding. This section is then picked up from the local latex processing centre or village headquarters and returned for central processing.

From these returns (which attempt to fill a critical feed-back gap which is almost universal) medium and higher level summaries are produced, giving progress at each level, giving statistics for higher levels, and giving exceptions for local extensionists (advisory staff), credit, input and marketing agents.

The subsystems dovetail within this process; thus delivery agents and stores receive lists of deliveries due and these lists are marked up to record actual quantities delivered. Marketing outputs are covered similarly. The Credit system then adjusts the farmer's account to reflect these transactions and others and deducts loan repayments or stop orders from the proceeds of sales.

2.4 Summary of the main features.

Space precludes a discussion of these facilities in detail, but summarising the main areas we have tried to cater for:-

2.4.1 Farmer Requirements: Attractive, profitable low-risk crops or livestock. Diversity is important to increase interest, smooth labour peaks and give a steady income. RISDA for example offers support for 19 main and 12 minor crops, and for 16 livestock species.

Appropriate technology and advice, incorporated in a viable extension package. (Manual systems tend to be generalised and necessarily vague about specific quantities or dates, or else unduly rigid).

Careful labour budgets. These exist as models for example in Kenya, but matching models to specifics is rarely done, and computer could provide a useful tool.

Secure supplies of inputs.
Secure market outlets and prices.
Straight forward farm accounts to reflect success or failure.

Credit and banking facilities.
Cheap and simple procedures operating in the vernacular.
Adequate provision for security of data and for privacy.

2.4.2 Extension Requirements: Extensionists (agricultural advisers) are the cornerstone of rural development. They carry similar responsibilities to those of the UK Agricultural Development Advisory Service (ADAS) but often have a wider remit, simultaneously to encourage and motivate, to monitor and report, to collect debts and to organise and deliver inputs. These mutually contradictory functions have to be carried out by each adviser covering on average 1000 to 1500 farmers. In practice they are compelled to attempt to contact perhaps the progressive farmers alone, supplemented by local meetings. At worst SCAPA should help them to do this in a more structured way, supplementing perhaps the World Bank's Benor Training and Visits system.[4] It should also reduce their office administration commitments by minimising the time (often 80%) spent compiling reports 'routine, ritual and unread'. At best it should provide rapid feed-back on the farmers who have problems, enabling the service to concentrate on those who need extension advice.

2.4.3 Support Requirements (Inputs, Credit, Marketing): These have been discussed already, but we would single out one aspect on credit as crucial, and this aspect is the capacity to leach out credit in small quanta only, relative to the farmer's achievements as reported by him and checked at key points by inspection. RISDA for example have a 17 stage package with four inspections in releasing grants or loans to farmers and this maps readily onto the SCAPA procedures. Given the high incidence generally of repayment defaults – 20% or more is common[6] – and the fact that the problem causing the default is often unrecoverable by the time manual systems have detected it – this approach has promise.

3 Technical features of the system's implementation

The system has been designed to meet general rural development requirements since experience to date has already shown that producing special purpose software for developing countries' requirements, results in expensive systems which are difficult to implement and develop and hence do not make the contribution to agriculture which computer systems undoubtedly can make. Some of the main features of SCAPA which assist this objective of wide application are as follows.

3.1 Hardware

A small ME29 mainframe is the current target machine, with connectivity and upwards-compatibility to larger systems so that spare capacity on these systems can be exploited. Dispersed mini and microcomputers, and remote terminals are an option, but are unrealistic in most cases until local power supplies, communication links and, most crucially, the availability of trained local staff, reach the necessary level.

Micros and minicomputers were immensely attractive, but we studied the requirements in detail, particularly ease of use by untrained staff, and found that key features – notably a powerful job control language and good data entry, data control and recovery procedures – were not generally available. To this should be added the advice frequently given in the computing press – e.g. by Herb Grosch or Alan Benjamin – that the best return from the very scarce computing expertise in developing countries would be gained by using medium sized mainframes. Hopefully this will improve in the next few years, but several other reports[8,9,10] suggest caution about the contribution micros can make in the immediate future except in a statistical or analytical role.

3.2 Software languages

All the 20 modules of the system are written in high-level languages; FORTRAN for the research oriented components and COBOL for the remainder. A key feature is the use of structured design techniques[7,16] to assist error-free implementation (claims elsewhere of a 50% improvement in this area seem justified in our experience) and more importantly, to assist modification and enhancements to meet very different local needs.

Structure

- **Modular** : **20 components**
- **Optional Subsystems** : **Input Supply**
  Credit Management
  Sales/Farm Accounting

Design Objectives

- Ease of use/reliability
- Transparent to particular application
- Administrative use
- Efficiency
- User enhancement
- Use for training purposes

Key Features

- Structured design
- High level languages
- Natural languages
- Data management/dictionary

Software Implementation

- Change control
- Validation
- Evaluation
- Field Trials
- General Release

Fig. 8 SCAPA system characteristics

3.3 Data Management

Conventional file systems, largely indexed sequential, are covered and recommended to first users, since to introduce higher-level data management systems in a large undertaking such as a development authority adds a significant level of complexity.

However, the merits of higher level systems such as IDMS (the Integrated Data Base Management System) are significant for experienced users, and transfer to this system for RISDA is planned. It is also provided for by some aspects of the implementation, for example, the use of data dictionaries. Efficient data handling is naturally a key area for most development authorities in view of the large transaction volumes which they encounter.

3.4 Natural Language Interface

The system interfaces with the real world through a message library. By substituting, for example, a Swahili or Bahasa Malaysia library for the standard English library, output in the local language is achieved. This capability is from our own experience surprisingly complicated (not complex) to impose retrospectively, and has, therefore, been monitored throughout implementation by maintaining versions in several languages.

3.5 Performance

Low cost and high throughput have been major objectives, and a target cost/farm/year of between £3-12 has been monitored against a particular benchmark.

4 Conclusion

SCAPA has now completed a year's trials in Kedah and Johor with RISDA, who are gradually extending its use to other states. It goes on general release from August 1982. The package's usefulness to cover the conventional planning recording and accounting requirements of estates and development authorities seems well established.[5]

Approaching the use of SCAPA as a means of *direct* interaction with the smallholders themselves requires caution and a healthy scepticism, although the response by smallholders in Malaysia has been good. The general obstacles mentioned at the outset – in particular the need for positive support by the farmers – require investigation, and a cost/benefit analysis under established procedures[11] against the requirements in each user situation. However, given this cautious background, the potential seems to many highly encouraging. To quote one of the European Development Fund chief agricultural advisors with whom it was discussed recently: 'The impedimenta of western agriculture – tractors and imported fertilisers and machinery – for us have little relevance; but the solutions to the problems of numbers, of a complexity and of a diversity; these are central'.

Acknowledgments

In the implementation team we have benefited immensely from advice and encouragement given by many people in government and institutes of development studies and similar organisations, particularly in Malaysia, Kenya, Zimbabwe, Nigeria, India, Zambia, Sudan, Eqypt, and in UK ODA, the World Bank and EEC.

We would like particularly to thank four people without whose support and encouragement the project would not have started or survived in a stringent commercial environment:
Professor Gordon Black, Director of Computation at UMIST;
Dr. Mohammed Nor, Director General of RISDA;
Doug Comish, Director of International Operations Division;
Bill Talbot, then Director of Research and Technology Division.

References

1 CHAMBERS, R.: 'Short cut methods in information gathering for rural development projects', IDS (sussex, UK) DP155, 1980.
2 GIMPEL, J., ELTON, J. and BAGHDADI, A.A.: 'Models for rural development', London, Acton Society Trust, 1979.
3 OKIGBO, BEDE, N.: 'Fitting research to farming systems – in policy and practice in rural development', ODI, London, 1976.
4 BENOR, D. and HARRISON, J.Q.: 'Agricultural extension – the training and visits system', World Bank, 1977.
5 TOTTLE, G.P. and ROBSON, F.: 'Computing requirements of development authorities in the smallholder sector, ODI Network paper 12, London 1981.
6 BATHRICK, D.D.: *Agricultural credit for small farm development*, Westvew Press, Boulder, Colorado, 1981.
7 JACKSON, M.P.: Principles of Program Design, APIC.
8 MACKEY, E.C.: 'Computing in programme planning and evaluation – experience in Malawi', ODI, 1981.
9 FELTON, M.W.: 'A review of experience with micros in evaluation and project planning in Nigeria', ODI, London, 1981.
10 LEAKEY, C.L.A.: 'The use of computer systems to aid production, management and administration', Proceedings of the Fourth Conference of the Association for the Advancement of Agricultural Sciences in Africa, Cairo, 1981.
11 PRICE GITTINGER, J.: 'Economic analysis of agricultural projects', World Bank 1972.
12 ROBSON, F.: 'Advanced Technology in developing agricultural economies', IFIP Conference on Informatics for Development, Cuba 1981.
13 DEBOECK, G. and KINSEY, B.: 'Managing information for rural development: lessons from Eastern Africa', World Bank staff working paper 379, 1980.
14 TRIANCE, J.M. and YOW, J.F.S.: 'Experience with schematic logic pre-processor', UMIST Manchester, 1978.
15 EVANS, K.L.: AFC credit management system, in computing for national development, p270 et f.f. British Computer Society, London, 1982.
16 VON PISCHKE, J.D.: When is smallholder credit necessary? Notes for the Economic Development Institute, World Bank, Washington D.C., 1978.

The PERQ workstation and the distributed computing environment

J.M. Loveluck
Science & Engineering Research Council, Rutherford Appleton Laboratory, Oxfordshire

Abstract

The paper is concerned with the PERQ single-user computer system, which originated in the Three Rivers Corporation of Pittsburgh, USA, and with its subsequent development in the UK. ICL has entered into an agreement with Three Rivers and now manufactures the PERQ in the UK and markets it in many countries, including the whole of Europe. The development is involving a collaborative partnership between Three Rivers, ICL, the UK Science and Engineering Research Council and Carnegie Mellon University, Pittsburgh.

The advent of relatively inexpensive, powerful, highly interactive single-user systems is likely to have a profound effect on computing methods in the next decade. The concept of *Distributed Interactive Computing* arises if such systems are interconnected by high-speed local and wide-area networks, which may also provide access to mass storage, high-quality print and graphics output devices and other more specialised facilities. The capabilities of the PERQ are summarised, emphasising those features which make it eminently suitable for the distributed interactive computing environment.

The Science and Engineering Research Council (SERC) has embarked on a major initiative to exploit this new technology and to provide a firm software base for this type of computing environment. An outline of the SERC Common Base Policy for single-user systems is presented, and the role of the PERQ as a component of this policy is indicated. The main threads of the SERC programme of hardware and software development within the framework of the Common Base Policy are described, emphasising a primary objective of the work, the realisation of the full potential of the PERQ within the distributed computing environment.

Some possible future trends in hardware and software advances in this field are mentioned. It is pointed out that the SERC software development programme for the PERQ permits a natural evolution towards a truly distributed computing system, in which the operating system as well as other resources is distributed over a network.

1 Introduction

It is almost a cliché to write of the enormous advances in computer power, coupled with increasing reliability and miniaturisation, which have taken place in the last two decades. In many respects the PERQ workstation is a continuation of this progression but we shall seek to demonstrate below that it also opens up the possibility of radical new departures in computing methods.

The dominant mode in which computers are used today, via a terminal connection to a remote central mainframe, is one which developed during the sixties, and is no longer necessarily the most appropriate to this decade. The dominance of large mainframes arose because, in the past, powerful processors were expensive and had to be shared. With the continual decrease in cost of CPU power it is no longer obvious that they represent the best solution for all computing needs.

One of the strengths of centralised computing facilities is that many users can share libraries of programs and databases, as well as expensive peripheral equipment such as high-quality graphics devices.

A single-user computer in isolation cannot be expected to provide all the resources which are normally made available to users of large mainframe computers. Indeed, for many applications these facilities are not all necessary. However, if a personal workstation, such as the PERQ, is connected to a high-speed local area network (LAN), giving access to some of the more specialised requirements, then one arrives at the attractive combination of the instant availability of a single-user system and the extensive resources of a large mainframe installation. These resources may be further enhanced by access to a wide area network (WAN).

It is this blending of powerful, highly interactive, single-user systems and high-speed networks which constitutes the Distributed Interactive Computing Environment. In the following Sections we shall examine the attributes of the PERQ which make it ideal for this sort of computing environment, the network connections available, and some trends in software and hardware developments which should bring significant future enhancements.

2 PERQ, ICL and the SERC Common Base Policy.

The Science and Engineering Research Council (SERC) has a long-standing interest in the PERQ. The Council's Rutherford Appleton Laboratory was in fact the first customer in the world to place an order for a system, in June 1979, shortly after the launching of the PERQ in the USA by the Three Rivers Computer Company (3RCC). It was clear to SERC at this time that the appearance of computer systems such as the PERQ would excite great interest in the UK university research community. The availability of low-cost single-user systems would have a significant impact on co-ordinated research projects, such as the Interactive Computing Facility (ICF), the Distributed Computing Systems Programme (DCS) and the Software Technology Initiative (STI). In addition, SERC was expecting to receive, through its sub-committees, numerous grant applications for single-user systems.

It was realised that there were a number of dangers inherent in this situation. In the first place, there was a risk that, by not having a suitable product available at the outset, UK industry would miss the opportunity to penetrate a potentially rich and rapidly expanding market. Secondly, it was inevitable that systems similar to the PERQ would be produced by other computer companies. It was likely that small companies would be involved, at least initially, and that they would provide minimal software support and field maintenance nationwide. As a consequence of this situation, it was envisaged that a number of different systems would be pur-

chased by SERC and its grant-holders, and this would result in considerable duplication of effort on basic software development.

The response of SERC to this predicament was twofold. Firstly, it formulated a policy, the Common Base Policy, to provide a network of single-user systems throughout British universities, with a common hardware and software base. Briefly, the Common Base Policy adopted by SERC consists of:

- A common software base, which uses the UNIX* operating system and includes Pascal (ISO standard) and Fortran 77 (ANSI standard) as programming languages. The Graphics Kernel Ssytem (GKS, which will be an ISO standard) has been selected as the basic graphics package.

- A common hardware base, which will consist of PERQ workstations connected locally by Cambridge Rings, with access to SERC's private X25 wide-area network (SERCnet) and British Telecom's public X25 network (PSS).

- Common access to special tools and facilities. For heavy computational tasks, vector and array processors should be available by network connections, which should also provide access to large databases and software packages, located at a single site. In addition, effective inter-user communications require integrated electronic mail and file transfer protocols.

The other aspect of the SERC initiative consisted of a proposal, made to ICL, that they should seek to negotiate an agreement with Three Rivers to market and manufacture the PERQ in the UK. In August 1981 ICL formally entered into commercial co-operation with the Three Rivers Computer Company, with an agreement which gave ICL marketing rights, not just in the UK, but in a number of other countries, including the whole of Europe.

As a result of this initiative there is now a strong collaborative partnership between 3RCC, ICL and SERC. This association has recently been extended to include Carnegie Mellon University (CMU) in Pittsburgh USA, which already had strong links with 3RCC. The latter collaboration has been a particularly fruitful one for SERC's Rutherford Appleton Laboratory (RAL); the implementation of UNIX on the PERQ (see below) by RAL relies heavily on the ACCENT operating system kernel developed by CMU. The RAL experience in using this software, in an environment for which it was not designed, has been invaluable to CMU in enhancing the robustness of their product.

In concluding this survey for the background to the launching of the ICL PERQ we note that the PERQ was not the first single-user system oriented towards the distributed interactive computing environment. The Xerox PARC Alto system[1], developed at the Xerox Palo Alto Research Centre (PARC) in the mid seventies, is especially pertinent to the development of the PERQ. Indeed, Brian Rosen, who played a major role in the development of the PERQ, was previously at Xerox PARC. The single-user PARC Alto systems were linked together to provide a

*UNIX is a Trademark of Bell Laboratories

powerful and flexible computing environment, which marked the way forward from time-shared systems.

The Xerox PARC Alto was a research project which was not commercially viable at the time, in large measure because it was not able to take advantage of recent technological innovations which have significantly reduced the costs of the necessary hardware components.

3 Technological progress in computer hardware

Before examining in detail the requirements of a single-user system in the Distributed Interactive Computing Environment, we mentioned some of the recent advances in computer hardware which have made it possible to satisfy such requirements in a system which can be marketed at an economically viable price (less than £25K).

Firstly, the cost of computer memory has declined steadily for many years. Significant cost reductions have coincided with the introduction of first 16K and then 64K RAM. The introduction of 256K RAM is now expected, and should be accompanied by further cost reduction.

The cost of computer processors also has declined with improvements in fabrication techniques of LSI chips. The introduction of bit-sliced processors has permitted the development of fast low-cost special-purpose processors.

There have also been considerable advances in bulk storage technology in recent years. Of particular relevance to single-user systems has been the introduction of Winchester discs, which provide reliable, low cost storage without special environmental conditions, a crucial factor for the single-user workstation.

A high-quality display is of paramount importance in the computing environment which we have described. Bit-map displays with a resolution of 1024 by 768 (A4 size) are now available commercially at reasonable cost (around £250). A flicker-free display is obtained by using a non-interlaced refresh, and the use of fast phosphors minimises smearing of the image as the display changes.

Finally, fast communications are crucial to the distributed computing environment. Special chips are now being produced for this purpose, which should greatly reduce the cost of communications interfaces while still permitting communication speeds in excess of 10Mbits/sec.

4 PERQ and the distributed interactive computing environment

The type of computing environment described in Section 1 realises its full potential only if the interconnected single-user systems satisfy certain criteria. In examining these requirements it should be borne in mind that a number of the services provided by central mainframe installations may be made available, in the distributed environment, by network connections. Resources which are shared by network users in this way are referred to as network 'servers' and their use is

illustrated in Fig. 1. The actual provision for any particular system will depend on local requirements.

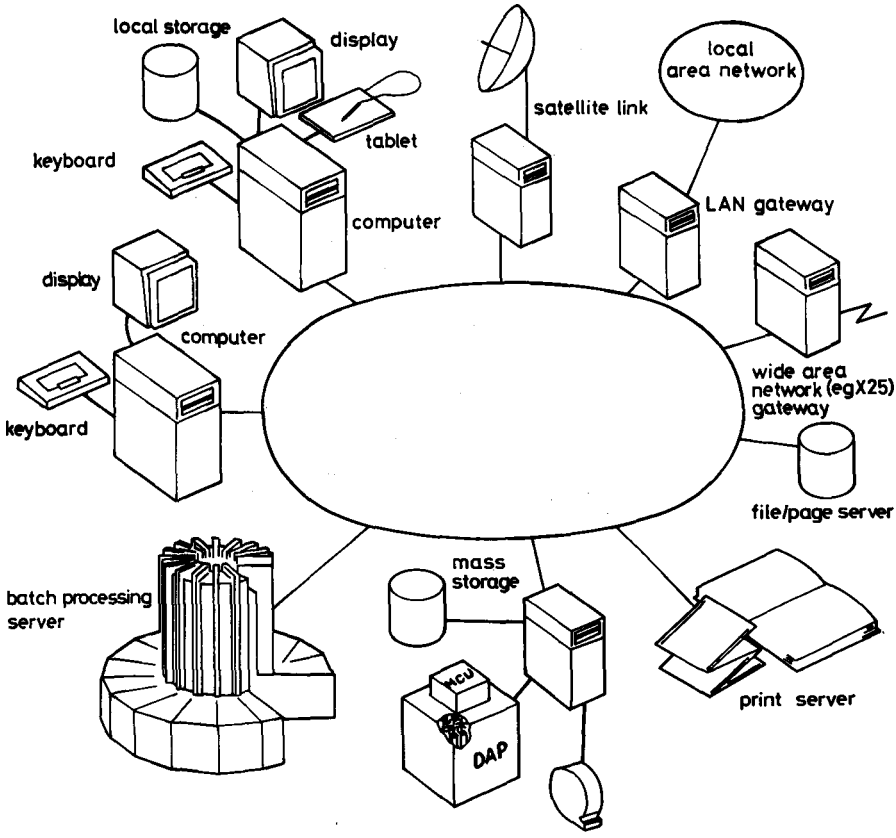


Fig. 1 Clients and servers on a Local Area Network

In this Section we outline the hardware capabilities of the PERQ workstation, emphasising their relation to the requirements for a single-user system (SUS) in the Distributed Interactive Computing Environment.

A general impression of the PERQ hardware is provided by the illustration in Fig. 2, and the different components are described in more detail below.

4.1 High-speed processor

The SUS should be adequate for all but the most demanding computing tasks, which may be performed by specialist network servers. The PERQ CPU executes approximately 1 million Q-codes (high-level machine codes) per second, corresponding to a processor power of two thirds of a VAX 11/780 in certain cases.

An additional aspect of the PERQ processor is that it is microprogrammed. This feature permits maximum flexibility for future developments, and for special applications, as well as yielding considerable speed increases when critical functions are microcoded. The microprogram resides in a Writable Control Store (WCS) consisting of 4k 48-bit words.

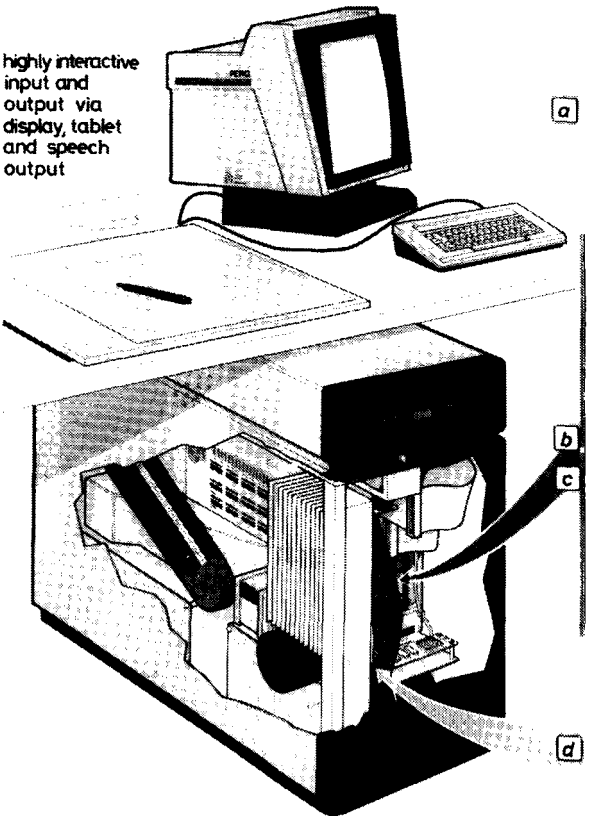


Fig. 2 PERQ hardware

a **Display.**
High quality, bit mapped A4 black and white display with P104 phosphor, flicker free, 60Hz non-interlaced scan, 768x1024 resolution, 64 bit parallel Raterop hardware.

b **CPU and Memory.**
High speed, user microprogrammable. 16 bit ALU up to 1 million Pascal P-codes per second. 4Kx48 bits of 170ns writeable control store.
256 Kbytes of 680 physical main memory. Large (32 bit) *virtual* memory.

c **I/O.**
Z80 controlling: RS 232 serial interface. IEEE 488 parallel interface. 10 MHz local network. Speech output.

d **Storage.**
Local filestore using 24 Mbyte Winchester disk with 97ms access time, 7Mbits/sec transfer rate 1Mbyte, double sided dual density, IBM compatible floppy disk

An outstanding feature of the PERQ is its special processor hardware and microcode, known as RasterOp, for performing logical operations on memory. This feature, which is described by Newman and Sproull[2] in their pioneering work on interactive computer graphics, is especially useful for the rapid manipulation of the display image. The PERQ Pascal compiler supports a RasterOp intrinsic, which invokes the RasterOp Q-Code, and software is provided to take advantage of this facility in the display of fonts and cursors. The RasterOp hardware performs shift, mask and merge operations on 64 bits at a time, which are pipelined into the hardware. The speed is limited by the memory cycle time of 680ns for 64 bits.

4.2 Large physical and virtual memory

The falling costs of computer memory have meant that single-user systems no longer suffer from significant restrictions on physical memory size. A growing computer user requirement is for a very large virtual address space, for example that spanned by 32-bit words (4 Gbytes). With the fast response of a single-user system, this is often a more important requirement than a large physical memory.

The PERQ provides options of 1/2 Mbyte or 1 Mbyte physical memory, and a 32-bit paged virtual memory. A 24 Mbyte Winchester disc provides extensive local filestore, and this is backed up by a 1 Mbyte floppy disc drive.

4.3 Fast communications

Local network communication, via a Cambridge Ring or Ethernet network, at speeds around 10 Mbits/sec are essential to the distributed computing environment. Access to wide area networks is also important, and can be provided by a ring gateway, by a dedicated hardware interface between a LAN and a WAN (see Fig. 1), or by direct links.

There are two possibilities for connection of the PERQ to a Cambridge Ring: a relatively inexpensive outboard hardware interface, or a more costly inboard one which could connect directly to the PERQ I/O bus and would, therefore, be considerably faster than the outboard unit.

The PERQ provides standard RS232 serial and IEEE488 (GPIB) parallel interfaces. The latter is particularly suitable for instrument control and data acquisition in a scientific environment. In addition, it can be used for communication with a local network, via a suitable outboard interface. An inboard Ethernet interface board is also available for the PERQ.

4.4 High quality display

An interactive environment presupposes high quality graphics with a display approaching the resolution of printed material. The PERQ has an A4 size, 1024 by 768 pixel black and white raster display, with 60Hz non-interlaced refresh. The display is bit-mapped in memory. Combined with the RasterOp hardware described above, the result is a high resolution display with very fast response. Hard copy output, with a variety of qualities, should also be available, but may be delegated to

network servers. However, plotters or matrix printers can be driven through the PERQ RS232 or GPIB I/O interfaces.

4.5 Efficient user interface

A flexible and versatile user interface is a necessity for an efficient interactive computing system. User input devices should be intimately related to the SUS and its display. The PERQ tablet, with its pointing device and buttons, provides a rich interface to the PERQ display. Hardware and software provisions permit an efficient utilisation of these tools, with dynamic, multi-window displays.

The PERQ hardware also contains the possibility of a less conventional user interface in the form of voice input and speech synthesiser.

5 SERC common base policy implementation for the PERQ

5.1 UNIX operating system

The current PERQ software consists of: the PERQ Operating System (POS); Pascal and Fortran compilers; a screen-oriented editor; a rudimentary debugger, and miscellaneous utility programs for file manipulation, etc. Although the PERQ Operating System makes very efficient use of the PERQ's unique features and is more than adequate for many applications, it has been designed specifically for the PERQ. As a consequence, it does not have the extensive supporting software libraries of more established products.

The UNIX operating system has been adopted as the SERC Common Base Policy standard because of its wide acceptance among computer users, especially in universities and research laboratories. There is an extensive set of UNIX utilities for the development, maintenance and documentation of computer programs, supported by active European and International user groups. An inherent flexibility is conferred upon the UNIX operating system by its modular structure: efficient tools for common tasks can be amalgamated, using the UNIX Shell language, to satisfy particular requirements. Furthermore, UNIX is a multi-process operating system, whereas POS is a single-process system and in this sense does not take advantange of the full capabilities of the PERQ. In addition, POS does not lend itself naturally to a more radical version of the distributed environment than that discussed above: one in which the operating system itself is distributed over a number of processors interconnected by local and/or wide-area networks. With such a system, not only special services but also processing power is distributed around the network. Whilst UNIX itself does not have such a capability, the PERQ implementation described below permits a natural extension to such a system, while maintaining full compatibility with standard Version 7 UNIX.

As part of the programme of collaboration between SERC and ICL, and in pursuance of the aims of the Common Base Policy, SERC is implementing, at its Rutherford Appleton Laboratory, an operating system for the PERQ which will

have the same system call interface as that of the Western Electric UNIX. The design aims are:

(i) A full Version 7 UNIX implementation.
(ii) 32-bit arithmetic, and paged 32-bit virtual addressing, with a flat address space.
(iii) An implementation which would permit a natural evolution to future enchancements, such as distributed operating systems.
(iv) The possibility of compiling and running programs developed under the existing POS system, where feasible.
(v) To satisfy a requirement of the SERC Common Base Policy: that all language implementations should be interworkable at the procedure call level. This requirement allows the possibility of mixed language programs, and obviates the necessity for translating sections of code from one language to another.

In order to realise these objectives the SERC implementation makes use of an operating system kernel called ACCENT, which has been developed by CMU as part of their SPICE (Scientific Personal Integrated Computing Environment) project in interactive computing.[3] The ACCENT kernel provides a suitable environment for a multiple process operating system, such as UNIX. The environment provided includes inter-process communication (IPC) via messages which are passed through logical entities called ports, allocated by the ACCENT kernel. Virtual memory management is also handled by the ACCENT kernel, and each process is given an independent virtual address space. This feature makes it impossible for processes to interfere with one another except by IPC, and makes for a very secure system. Additional protection in the ACCENT kernel obtains from the fact that ports are protected kernel objects, and the kernel provides processes with a secure *capability* to send a message to, or receive a message from, a port. Port access rights cannot be forged or accidentally created, and this means that the areas of interaction between different processes can be delineated precisely. Process management, including process creation, destruction and UNIX 'fork', are also provided. The ACCENT kernel requires a 'microkernel' of microcode support, which provides process queue management, low-level scheduling, I/O support, interrupt handling and virtual memory address translation and paging.

Building UNIX on the ACCENT kernel automatically satisified a number of the requirements outlined above. However, it did carry the disadvantage that the Unix implementation would be dependent on software which was still under development, and outside SERC control. The lack of stability of the ACCENT kernel has indeed been a major problem in this implementation. A further difficulty, which was not foreseen, has been that UNIX itself has been found to contain bugs which only manifest themselves when the system is ported to another machine. In fact, UNIX has been found to be much less portable than is generally supposed.

A major part of the UNIX operating system and utilities are written in the C programming language, so implementation of a C compiler is essential to PERQ UNIX. There exists a portable 2-pass version of the C compiler[4], which is supposed to separate the machine-dependent parts into the second pass. In fact, the separa-

protocols. Until the ISO standard protocols have been defined, these protocols are expected to be extensively used, and they should also influence eventual ISO standards. The protocols defined by study groups of the Data Communications Protocol Unit have been published in a series of documents which have come to be known collectively as the 'Rainbow Books'. The 'Yellow Book'[6] in this series contains the definition of a Transport Service (TS) Protocol. Transport Service provides processes which wish to communicate with an interface which is consistent across all underlying networks, in such a way that users are insulated from the peculiarities of particular networks and their low-level protocols. SERC intend to provide a Yellow Book Transport Service over Cambridge Rings, and this will be implemented as Transport Service Byte Stream Protocol (TSBSP). This software presents the same transport service to user programs as the wide area network X25 software which will be used by SERC, and will result in a transparent extension from local to wide area network.

The TSBSP protocol requires a basic block interface to the Cambridge Ring. Such an interface has been written, to run under POS, for the PERQ GPIB/Cambridge Ring interface. A version which will run under UNIX is now being devised.

5.3 Graphics

The combination of a high-quality display and the special RasterOp hardware of the PERQ constitute a powerful graphics capability, and it is clear that a large number of applications of the PERQ will exploit this potential.

The RasterOp intrinsic which, as mentioned above, can be called from user programs, is a powerful tool for raster graphics applications. In addition, POS provides facilities for line drawing (supported by microcode) and for the subdivision of the display into a number of different areas, or windows. The latter facility permits the display of text or graphical information in a designated window, and provision is made for the creation, deletion and modification of the size and position of these windows. Using the RasterOp procedure, it is possible to develop an enhancement which allows windows to appear to move over each other, while preserving the display beneath. An algorithm which illustrates the use of RasterOp for this purpose is presented in an Appendix. The use of windows permits the user to selectively display items from a considerable quantity of textual and graphical information, in much the same way as one would select from material laid out on an office desk.

Extending these facilities for the management of windows to a multiprocess operating system such as UNIX presents a number of problems. Clearly, it would be desirable for each process to be able to create and modify one or more windows, but it may also be desirable for a number of processes to display information in the same window in a co-operative way, to share I/O devices between windows belonging to different processes, and for 'hidden' windows to signal essential information to the user. In order to avoid conflicts between the requirements of different processes, a special 'window manager' process may be necessary to supervise the creation and movement of windows.

The graphics primitives available on the PERQ are evidently very powerful tools, but there is also a need for a graphics applications package. A basic graphics package for the PERQ is being developed by ICL. In addition, ICL and SERC are collaborating on the implementation of the Graphics Kernel System (GKS), version 7.0, on the PERQ. The Rutherford Appleton Laboratory is currently installing an earlier version of GKS (Version 6.2) which will be ported to the PERQ as soon as full UNIX facilities are available.

The Graphics Kernel System has very recently been adopted as an ISO Draft International Standard, with a large measure of international support. In a historical account of the development of the standard,[7] which contains the draft Proposal as an appendix, it is argued that the timing of developments in graphics hardware played an important role in delaying the introduction of a graphics standard, and it was not until 1974, at an IFIP WG5.2 meeting in Malmo, Sweden, that any significant steps were taken towards the definition of a standard. As a result of an initiative taken at this meeting, a workshop (Seillac I) was organised by IFIP WG5.2 at Seillac, France, in 1976, which laid the seeds for the development of the GKS standard. The use of virtual input and output devices was accepted at Seillac I, and some appropriate output primitives and input devices were defined. A major clarification was the distinction between *viewing* a picture, and *modelling* a picture out of smaller items. In addition, it was agreed that an initial goal should be to define a core graphics system for viewing aspects of a picture already constructed in world co-ordinates.

In 1977 an ISO working group was set up to consider computer graphics standardisation, and the current GKS Draft Standard is the result of a series of meetings of the group in the period 1977-1982.

A major objective of GKS was to allow easy portability of graphics systems between different installations. In addition, the Standard is not specific to a particular language and it should be possible to implement it in any of the ISO standard programming languages. A central concept is that of a graphics workstation, which provides the logical interface through which the applications program controls physical devices. Different types of workstation with different capabilities, reflecting the available hardware, are recognised. The concept of virtual input and output devices has been retained in GKS, but output primitives have been refined by allowing them different attributes. GKS allows great flexibility in the choice of viewpoint, and this is achieved by defining three different co-ordinate systems (world co-ordinates, normalised device co-ordinates and device co-ordinates) and two distinct types of transformation. Finally, the use of device-independent segment storage admits the possibility of later display on devices associated with workstations not initially activated. This provision is useful if, for example, a user wishes to obtain plotter output of a picture, parts or all of which have been previously viewed on a VDU.

6 PERQ hardware enhancements

A number of hardware enhancements to the PERQ have already been made since delivery of the first systems. These include extended memory (1/2 and 1 Mbyte

systems instead of the original 1/4 Mbyte) and an Ethernet controller board. Further enhancements which are expected in the near future are a 16k Writable Control Store, an improved I/O board (faster and with additional facilities) and a Versatek plotter-printer interface.

While PERQ represents a significant advance in the capabilities of single-user computers, further refinements will undoubtedly appear in the near future. Speculation on the desirability of various hardware enhancements to the PERQ is inevitably somewhat subjective, since each user will have his or her own particular requirements. Nevertheless, a number of possible hardware developments may be selected which would have applications in several areas.

Clearly, it is important to maintain the momentum of architectural innovation found in the PERQ. One of the major strengths of the PERQ is that it is micro-programmable, and the extension of the WCS to 16k words will allow a much richer instruction set and opens up a wide range of potential applications. The performance of the PERQ would also be greatly enhanced by the addition of paging and floating point hardware and fast cache memory, as well as increased main memory. For many graphics applications hardware for clipping and for three-dimensional rotations is also desirable.

The PERQ display could be enriched in a number of different ways. A colour display is an obvious attraction, but with current technology such a device would have much lower resolution than the present black and white display. A grey level display, using several bits per pixel, has the effects of a higher resolution while maintaining a greater flexibility. For many applications larger displays are desirable, and two possibilities are an A3 size landscape display with a resolution of 1024 x 768, and a dual A4 head display which would maintain the present resolution of the PERQ.

The user interface of the PERQ could be greatly enriched; for example, by developing the voice I/O possibilities, and by adding further input devices, such as a mouse and a touch-pad.

Finally, the I/O connectivity of the PERQ could be increased by addition of further serial ports and by the addition of a multibus parallel port interface, which is used by many devices and scientific instruments. Use of the PERQ for real-time control and data acquisition will require a significant increase in I/O rates, either by improvements to the present Z80 controlled I/O or by direct interface to the PERQ I/O bus.

7 Conclusions

We have summarised the capabilities of the PERQ personal workstation, emphasising those aspects which make it eminently suitable for distributed interactive computing. This environment, where powerful, highly interactive single-user systems with high-quality displays are interconnected by high-speed networks, is expected to have a profound influence on computing methods in the next decade.

The UK Science and Engineering Research Council is one of the biggest users of computing resources in the UK, and has anticipated this transformation in computing methods by developing a co-ordinating plan, the SERC Common Base Policy, which is aimed at avoiding wasteful duplication of effort in the development of basic software for single-user systems, and providing the hardware and software technology to enable an efficient utilisation of such resources. The plan envisages that by 1983 SERC will have over 200 PERQs distributed throughout the UK at universities and SERC laboratories, and that these systems will be linked to each other, and to existing facilities, by local and wide area networks.

The SERC Common Base Policy is not intended as a fixed standard but is expected to evolve in an orderly way, to reflect continuing technological innovation and advances in computing methods. In particular, the groundwork has already been laid for a transition to computing systems in which the operating system, as well as computing facilities supplied by network servers, is distributed over a network. This change is expected to take place around the end of this decade.

Acknowledgments

I am grateful to Bob Hopgood and Rob Witty for comments on earlier drafts of this paper, and for providing unpublished material on the background to SERC's involvement with the PERQ and the formulation of the SERC Common Base Policy. Dale Sutcliffe provided helpful comments on the GKS Draft International Standard.

The SERC software development work reported in this paper is being done within the Distributed Interactive Computing Section at SERC's Rutherford Appleton Laboratory, and I should like to thank all my colleagues in this section for useful discussions and access to unpublished documentation. Specifically, Len Ford, Liz Fielding, Janet Malone, Colin Prosser, Trudy Watson and Tony Williams have worked on UNIX implementation for the PERQ, and Chris Webb has been concerned with installing GKS. I have also benefited from discussions with Keith Fermor, Bill Sharp and Chris Wadsworth, who are concerned with communications.

References

1 THACKER, C.P., McCREIGHT, E.M., LAMPSON, B.W., SPROULL, R.F., and BOGGS, D.R.: 'Alto: a personal computer' *Computer structures: readings and examples* (SIEWIORECK, D., BELL, C.G., and NEWEL, A., Eds.) McGraw-Hill, 1980.

2 NEWMAN, W.M., and SPROULL, R.F.: *Principles of interactive computer graphics*, McGraw-Hill, 1973.

3 RASHID, R.F.: 'ACCENT: A communication oriented network operating system kernel', ACM SIGOPS, 1981, 15 (5), pp. 64-75.

4 JOHNSON, S.C.: 'A tour through the portable C compiler', UNIX programmer's manual, Bell Laboratories, 1979.

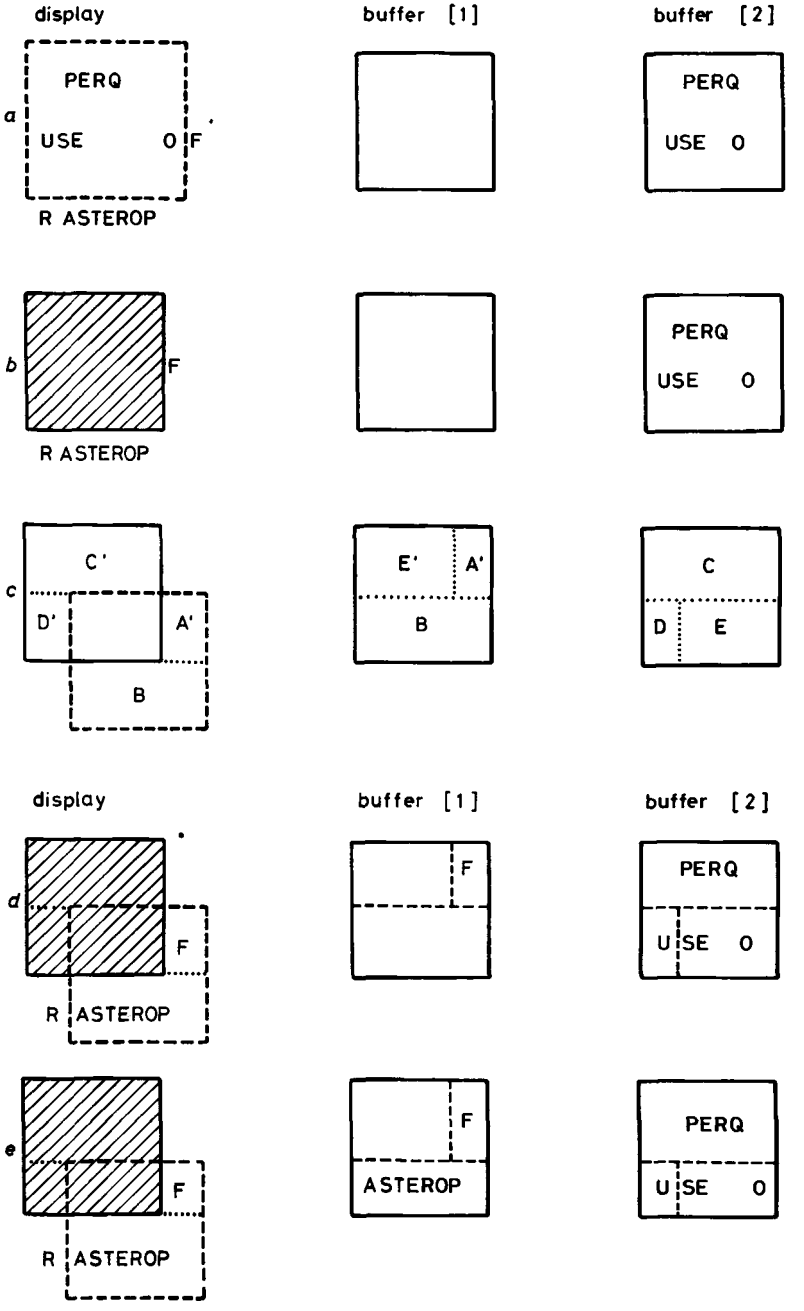
5 FELDMAN, S.I. and WEINBERGER, P.J.: 'A portable Fortran 77 compiler', UNIX programmer's manual, Bell Laboratories, 1979.

6 Yellow Book: 'A network independent transport service', prepared by Study Group 3 of the Post Office PSS User Forum, SG3/CP(80) 1980-2-16, 1980.

7 HOPGOOD, F.R.A.: 'GKS - The First Graphics Standard', Rutherford Appleton Laboratory report RL-82-007, 1982.

Appendix

In this appendix we present an algorithm for moving an arbitrary rectangular area, or 'window', over an arbitrary display, using the facilities available from the PERQ



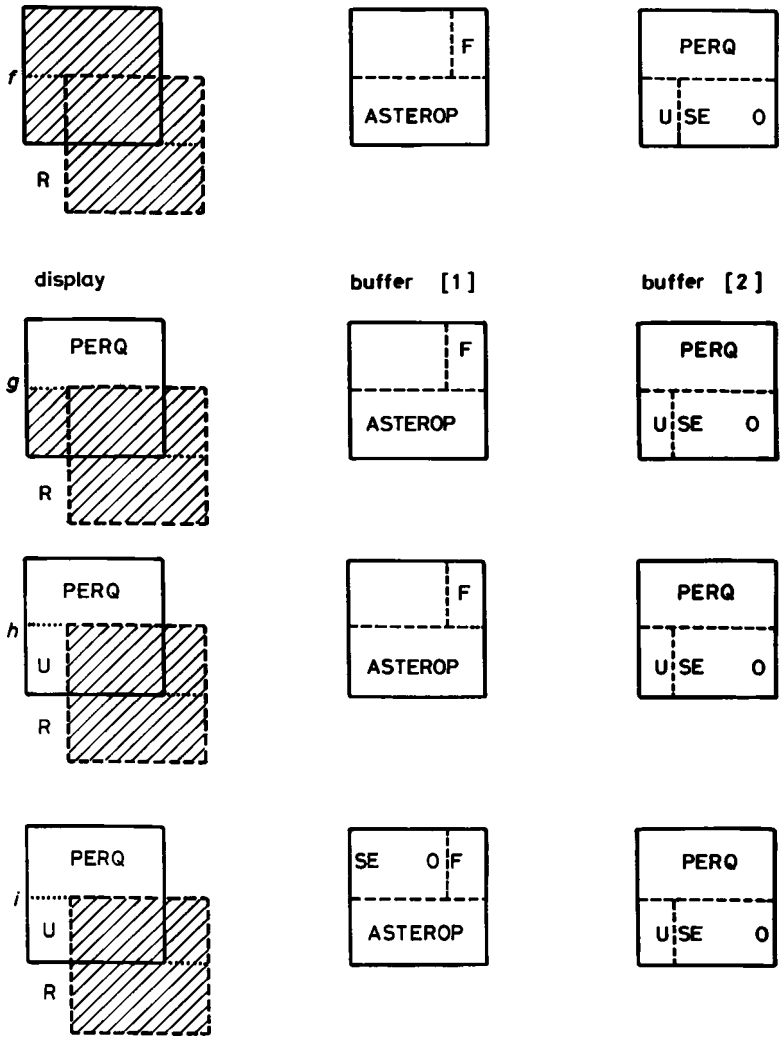


Fig. 6 The use of RasterOp for moving 'windows' over an arbitrary display is illustrated.

(*a*) Step 1. The part of the display bit-map to be covered by the window is copied into buffer[2].
(*b*) Step 2. The window is copied onto the screen; the window display is shown hatched, but may have arbitrary content.
(*c*) The rectangular areas involved in later copy operations are shown.
(*d*) Step 3. Area A is copied from the display bit-map to buffer[1].
(*e*) Step 4. Area B is copied from the display bit-map to buffer[1].
(*f*) Step 5. The window is copied to the new position in the display bit-map.
(*g*) Step 6. Area C is copied from buffer[2] to display bit-map.
(*h*) Step 7. Area D is copied from buffer[2] to display bit-map.
(*i*) Step 8. Area E is copied from buffer[2] to buffer[1].

RasterOp procedure and its corresponding hardware. The various steps of the algorithm involve copying from the display bit-map to (another part of) itself, from the display bit-map to memory buffers and vice versa, and between two memory buffers. All of these copy operations may be performed by RasterOp procedure calls using the RRpl (RasterOp Replace) function, which replaces the destination bit-map with the source bit-map. Two buffers are used alternately to store bit-maps of the part of the display hidden by the window, and in this way the original display is restored as the window passes over. The algorithm is presented as a 'pseudo program', and the steps are illustrated in Fig. 6. Figs. 6*a* and *b* show the initialisation stages, steps 1 and 2, and Fig. 6*c* indicates the rectangular areas which are referred to in later steps; a prime is used to indicate the destination area. Fig. 6*d* to *i* illustrate the steps (3 to 8) required to actually move the window from one position to another, and to update the buffers appropriately. The figures show an initial movement of the window, but these steps can be repeated for further movement.

```
BEGIN
  { Initialisation }
  create buffer [1], buffer [2];
  index1:=1; index 2:=2;
  (1) buffer [2] gets part of display to be concealed {6a} :
  (2) display bit-map gets window {6b} ;
  { this may be copied from another part of
  { memory, or constructed directly by Raster Op }
  REPEAT
    calculate the origins, widths and heights of rectangular areas to be moved;
    { save in buffer[index1] the parts of display
    { that are going to be covered up }
    (3) buffer [index1] gets area A from display bit-map {6d} ;
    (4) buffer [index1] gets area B from display bit-map {6e} ;
      now the window is copied in the new position
    (5) display bit-map gets window at new position {6f} ;
    { re-draw the bits of display which have been
    { uncovered, using the display saved }
    (6) display bit-map gets area C from buffer[index2] {6g} ;
    (7) display bit-map gets area D from buffer [index2]{6h} ;
    { complete the image under the window in
    { buffer[index1], by copying from buffer[index2] }
    (8) buffer[index1] gets area E from buffer[index2] {6i} ;
    { now the role of the buffers is swapped, ready for
    { a succeeding move }
    index1:=3-index1;
    index2:=3-index2
  UNTIL finished;
END;
```

Some techniques for handling encipherment keys

R.W. Jones
ICL Product line Planning Division (Systems Standards Sector), Bracknell, Berkshire

Abstract

While modern methods for encipherment can provide very strong protection of the privacy and security of information, they rely ultimately on the protection of the encipherment/decipherment keys from unauthorised access and use. The paper describes the author's work on a new technique for the handling of these keys. It gives the user greater flexibility than other methods that have been described, and enables him to define complex key manipulations clearly, and to be assured of their security because of their construction from basic secure operations.

The essential feature is that information is added to a key to define its permitted use – for example, whether it may be used for encipherment or for decipherment and whether it is to operate on data or on another key; and the resulting combination is enciphered as a single item. By this means a key can be confined to a single specified use. The principle can be applied also to provide a form of public key system. It could influence both the notation used to describe encipherment/decipherment key manipulation and the standards for data encipherment that are now emerging.

1 Introduction and background

It is becoming increasingly important for a data-processing system to provide an environment in which a user's information is not only secure in the sense that accidental corruption or deletion is very unlikely, but also in the sense that it is very unlikely that a deliberate intelligent unauthorised attempt to read or corrupt information will succeed. If data is kept and processed at one installation physical security (locking doors etc.) and organisational checks (e.g. devoting computing resources to only selected tasks when sensitive data is handled and carrying out manual checks at the beginning and end of such handling) may provide much of the needed security. If an installation is required to handle concurrently a mixture of jobs with mutually private data then the computing equipment itself must provide security features to enforce segregation of users and their data. Encipherment is one of the most powerful of the techniques available and becomes obligatory whenever users' data is exposed on a publicly accessible medium such as a communication link. It has also the advantage that the mechanism involved can be isolated as one unit.

This paper assumes that a data-processing installation should provide an encipherment and decipherment function to protect data both stored (for example on disc) and when in transit. Such a function can also be used to provide user authentication.

The standards which are being developed for the use of encipherment of data prescribe a standard algorithm, realised in hardware or software, and a particular key which is kept secret from the unauthorised. The currently best known algorithm is DES, described in Reference 2. A mechanism conforming to DES has two modes of working; one for encipherment and one for decipherment. In each of these modes it has two inputs, a 64-bit data item and a 56-bit key, and one output, a 64-bit data item. When enciphering the input data item is clear text and the output data item is cipher text. When deciphering the converse is true. Moreover the same key must be used when enciphering and deciphering. The 56-bit key is held in a 64-bit item, 8 bits of which are not used for encipherment.

2 Data key protection

A simplistic approach to encipherment, using such a function at a computer installation, would be to allow a user to type in a key, to store the key in part of the computer store dedicated to his use and to present that key to the enciphering device as required by his directions. Similarly, if the user were using a terminal with enciphering capability remote from the computer he might type in the key which would then be presented to the attached enciphering device.

Working thus, the key may be seen by the unauthorised either because the authorised user is careless with it, or because the mechanisms which should protect it in the computer store contain flaws. Reference 1 describes a more secure method of handling keys in which only a single key, the master key, is held in clear form at any one installation. Other keys are themselves enciphered using the master key or a derivative of it or another key which is itself enciphered. Now, if the master key is kept in a secure module in which the encipherment/decipherment algorithm is also available, all other keys may be supplied to that module in enciphered form. If we then suppose that the external (enciphered) form of a data-enciphering key is obtained by an unauthorised person he cannot use it without at least gaining access to the computing facility whose master key has enciphered it. This principle, that no key (with the possible exception of the master key in very tightly controlled circumstances) should ever appear in clear outside the encipherment module, is adopted here. Users still depend upon access controls in the operating system to keep their enciphered keys safe from other authorised users of the encipherment modules; for a discussion of such controls see, for example, Reference 5.

Reference 8 imposes further restraints on each user of the encipherment module which effectively force him to use his own version of the most fundamental key-enciphering key. This is considered further in Section 7 of this paper.

3 Key hierarchies

Given an installation master key we have a hierarchy of information. The master key enciphers data-enciphering keys (DEKs) which in their turn encipher data. Reference 1 points out that at least one more level in this hierarchy is useful. If two locations communicate by means of a telecommunication link a common session key (KS) is needed to encipher data sent by either party during a session. It may be generated by one of the parties and sent to the other. During the session

KS will be held at each location, enciphered by the master key of that particular location. The location which generates KS must send it securely to the other. If the two locations have the same master key it may be used to encipher the session key in transit, but this is undesirable since it increases the use and therefore the potential exposure of the master key. If the location generating KS is in control of the other and is considered more secure it may itself store the other's master key and use it to encipher KS (see Fig. 1). Alternatively, if there is no such subsidiary relationship, the two locations may share a key which is restricted to protecting session keys when they are sent from one to the other (see Fig. 2). In either case we now need to store securely the key-enciphering key (KEK) which is to encipher KS in transit and we need to transform KS from its encipherment by the master key to its encipherment by the transport key. This is illustrated in Fig. 6 which shows a key-enciphering key KK enciphered, when held outside the secure module, by the master key: the notation $E_{KM}KK$ is used to mean the value of KK enciphered by the value of KM. It is read into the secure module where it is deciphered. The key K, which is to be sent elsewhere, is also enciphered by the master key when held outside the secure module. It is read into the secure module and deciphered. KK is then used to encipher K and the enciphered form is output and sent to the location where KK is known. This procedure has a possible flaw which is discussed later.

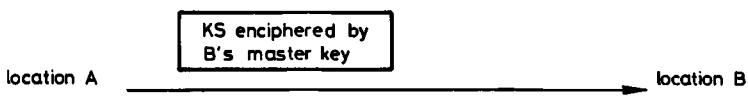


Fig. 1 Session key distribution using location master key

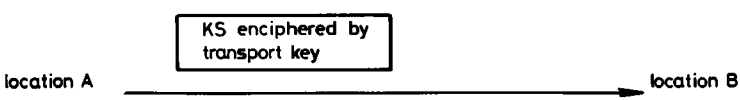


Fig. 2 Session key distribution using transport key

An extra level of enciphered keys is also needed when enciphering stored files. A simple implementation would encipher a file using a data key and store that data key, enciphered by the master key. The data key may need to be re-enciphered either when the master key changes or when the file is sent to another location which has its own master key. That re-encipherment poses the same problem as the transfer of the session key described above. Deeper hierarchies are also useful. As an example, a particular user may have the authority to access the files of a number of other selected users. If the data key of each file is enciphered by a key known only to the senior user and stored with the file he has a skeleton key to the files.

The technique can be applied to deeper hierarchies of authority as needed. This is illustrated in Fig. 3. The boxes at the bottom of the diagram represent the files. E_{K1}file1, for example, is the file called file1 whose data is enciphered by the key K1. A user who is allowed to access that data is given the key K1 which he must keep securely, for example by keeping it in a part of the filestore to which only he has access, enciphered by the master key. This is not illustrated in the diagram.

The boxes in the middle row of the diagram represent records which contain the file keys. The left hand box, for example, represents a record which contains K1 and K2, both enciphered by a KEK called KKA. A user who is allowed to access the data of file1 and file2 is given the key KKA which he must keep securely, for example in his private store enciphered by the master key. The arrow in the diagram from $E_{KKA}K1$ to E_{K1}file1 means that the former gives access to the latter.

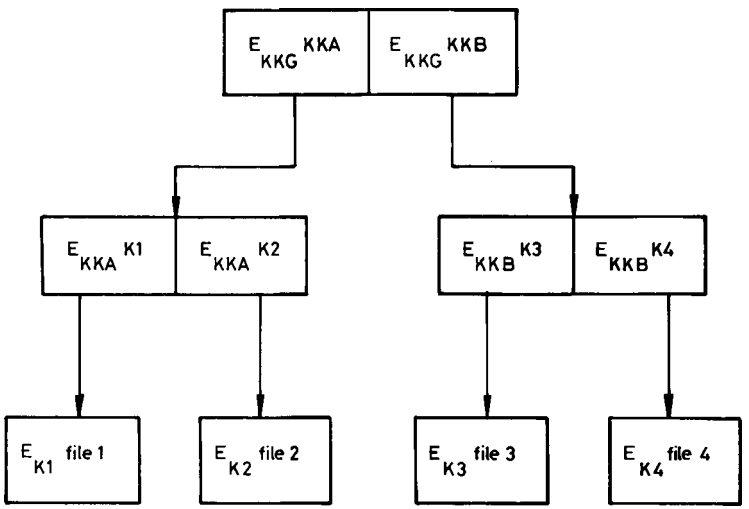


Fig. 3 Use of a hierarchy of encryption keys

Similarly the box at the top of the diagram represents a record which contains enciphered keys which together give access to all the files shown.

The hierarchy can, of course, be avoided by distributing K1, K2, K3 and K4 appropriately to the users. The advantage comes when there are many files in a group and many users. When transferring to a new master key any keys enciphered by the master key must be changed securely, as explained earlier. The files themselves and keys enciphered by other KEKs are not changed.

4 Types of key

The introduction of this extra level of key-enciphering key into the hierarchy itself introduces a security problem. We wish to make sure that it is not possible for an authorised user, by accident or otherwise, to cause a key to be output from the secure module in clear form. Therefore, he is not provided with access to the primitive encipherment and decipherment functions, but rather with functions which enable him to use them but prevent this misuse. Some operations he needs to perform are shown in Figs. 4, 5 and 6. Fig. 4 shows the operation of data encipherment. $E_{KM}K$ is the data key enciphered by the master key. KM is the master key which is input to the decipher operation from inside the secure module whose boundary is represented by the dotted line. The output from the decipher operation is the clear form of the data key, K, subsequently used to convert the clear form of

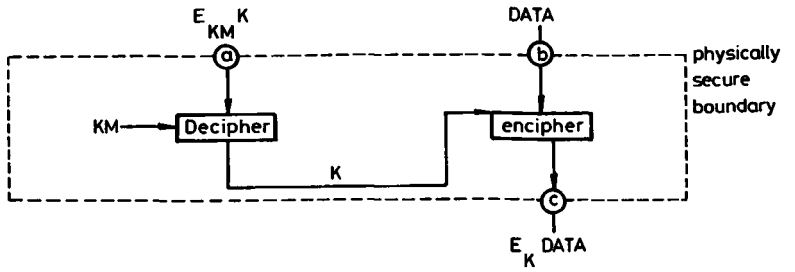


Fig. 4 Date encipherment

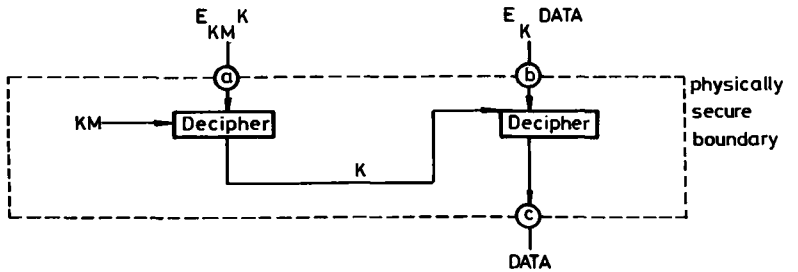


Fig. 5 Data decipherment

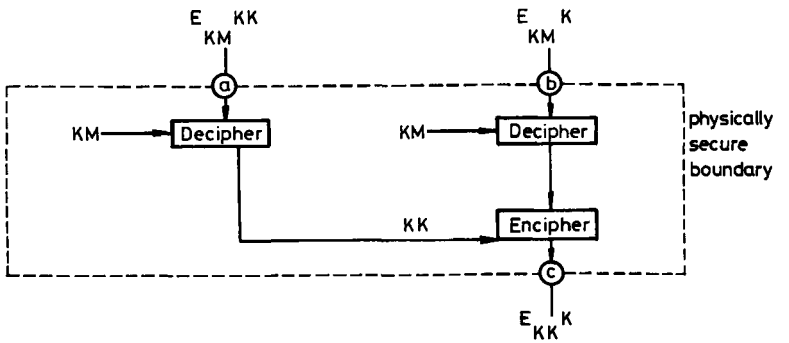


Fig. 6 Re-encipherment from the master key

the data item, DATA, to the enciphered form, E_KDATA. Fig. 5 shows the corresponding data deciphering operation. Fig. 6 shows the re-enciphering operation needed to convert a key enciphered by the master key, $E_{KM}K$, to its encipherment by another key-enciphering key, $E_{KK}K$. This entails the production of the clear form of the key-enciphering key, KK, inside the secure module from the stored form which is enciphered by the master key. Now all three of these operations and at least one other (data key re-encipherment to the master key) need to be provided by the secure module to authorised users. A possibility is that the user have operations which the physically secure module provides, corresponding to these four operations, with input and output parameters represented by (a), (b) and (c)

in each case. However, without further precautions the authorised user then has operations available to him to obtain a key in clear outside the secure module. This is illustrated in Fig. 7. Reference 1 overcomes this problem by ensuring that a key-enciphering key is not enciphered, at an installation, by the master key used to encipher data keys but by a variant of the master key reserved for key-enciphering keys. The operations to encipher data (Fig. 4) and decipher data (Fig. 5) assume that the key involved is enciphered by the original form of the master key (KMO). The operation to re-encipher a key from the master key to another KEK assumes that the KEK which is to encipher the data key is itself enciphered by a variant of the master key. Thus if the data key is subsequently offered as data input to the decipher operation, as in Fig. 7, the correct clear output will not be produced because the KEK will be wrongly deciphered.

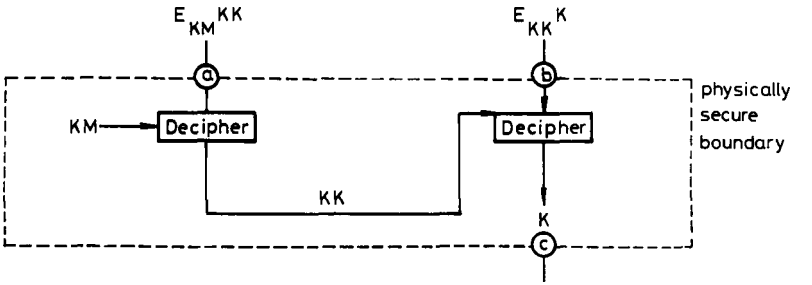


Fig. 7 Decipherment of a key

The rest of this Section describes a different technique for key manipulation and is the chief contribution of this paper. There is only one master key per installation, but information about any key is stored with the key when it is enciphered. This information when revealed controls the exact meaning of the operation called for using the key. This gives the user both greater flexibility in the operations he can perform and a convenient way of using a deeper hierarchy of keys. Fig. 8 illustrates this. Assume that the key, as in DES, is 56 bits long and is held in a 64 bit item. Assume moreover that the extra 8 bits are available – e.g. they are not required to record parity. Fig. 8 shows one of the eight bits used to record whether the item holds a data key or a key enciphering key (the eight bits are shown together in the illustration for convenience; in fact, they are scattered throughout the 64-bit item). Once the item is enciphered this information cannot be changed in a controlled fashion without correct decipherment.

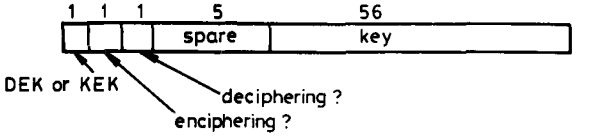


Fig. 8 Type information stored with the key

Now consider the operations which the secure module supplies to its authorised users. The logic which controls the operations of the encryption module treats each item as having a type in the sense in which this term is used in programming

languages. We may compare it to a processor performing arithmetic on typed items. In interpreting 'A+B' the arithmetic processor takes account of the types of A and B to decide the type and value of the result. If both have type 'integer' the result has type 'integer'. If both have type 'real' or one has type 'real' and one 'integer' the result has type 'real'. If either has type 'alphabetic' the operation is illegal. In the present case the two principle types are 'word' and 'key' and they are used to decide the meaning of the legal operations. A 'key' is a *clear* key and is further subdivided into KEK and DEK. A 'word' is an item whose value is not a clear key. It may, therefore, in practice, be an unenciphered data item, an enciphered data item or the encipherment of a key and the bits which show how it is to be used, as just described. An item read into the encryption module is given the type 'word'. An item with the type 'key' is produced by decipherment, using a KEK, a particular case of this being decipherment by KM. The subsidiary information which marks it as a DEK or KEK is revealed when the item is deciphered (see Fig. 8). It is possible to perform the operation 'decipher', using a KEK, upon an item which is not an enciphered key. The result is meaningless and does not endanger security. The logic of the module is such that any attempt to output an item of type 'key' will cause its type information to be stored with it and the resulting item to be enciphered by KM before output.

We may return now to consider some of the operations the user wishes to perform. To decipher data (Fig. 5) a program in the secure module may cause a 'word' to be deciphered by KM to reveal a DEK. The revealed DEK can then be legally used to decipher a 'word' (the enciphered data) and produce a 'word' (the clear form of the data). To re-encipher a session key from its encipherment under KM to the master key of another location (Fig. 6) a 'word' (the master key of the other location enciphered by KM) is deciphered by KM and revealed as a KEK. A second 'word' (the session key enciphered by KM) is deciphered by KM and then re-enciphered by the KEK to produce a 'word' (the session key enciphered for the other location). Any attempt to output the clear form of the key is foiled since the mechanism refuses to output an item of type 'key' without enciphering it.

Table 1. Encipherment and decipherment functions; effect of parameter types

	Function	First parameter type	Second parameter type	Result type
1	encipher	DEK	word	word
2	encipher	KEK	key	word
3	decipher	DEK	word	word
4	decipher	KEK	word	key

The module provides its users with encipher and decipher functions based on more primitive operations (such as DES, See Reference 2) but taking into account the types of the parameters. Table 1 lists the legal parameter combinations. In each case the first parameter is the key and the second is the item to be enciphered or deciphered. The type of the first parameter must be 'key'. The following notes describe the meanings of the combinations listed.

Case 1 is used to encipher data. The item specified by the second parameter is enciphered by the key specified by the first parameter to produce an item of type 'word'.

Case 2 is used to encipher a key. The KEK specified by the first parameter enciphers the key specified by the second parameter, together with its usage information as shown in Fig. 8.

Case 3 is used to decipher data and is the converse of Case 1.

Case 4 is used to decipher a key to produce its clear form and its type information and is the converse of Case 2.

A function 'reveal' is also needed which deciphers its operand, using KM. In some cases it would be possible to give useful secure meanings to parameter combinations of 'encipher' and 'decipher' not shown in Table 1 by incorporating implicit 'reveal' operations. This is an implementation matter not considered further here. 'Reveal' is sometimes implicit in the notation introduced in the next Section.

A 'store' operation places the value of its operand in store, where it may be output from the secure module. If the operand has the type 'key' (i.e. KEK or DEK) it is enciphered by KM and given the type 'word' before being stored.

5 Notation

The operations described in Section 4 are the basis of the system of typed keys and, together with some extra storage in the secure module, enable a secure hierarchy of keys to be used whose depth depends only on the amount of store. The following examples introduce a notation for programming the secure module (from its user's point of view) and illustrate the variety of operations which can be performed securely. The examples are concerned only with enciphering and deciphering single data items and manipulating keys and concentrate particularly upon the ability to use an indefinite hierarchy of keys. With some additions it is possible to describe more complex operations, for example the modes of operation of encipherment described in Reference 3. Interpretation of the notation by the logic of the secure module is described in Section 8.

1 E_KDATA := K enc DATA:

enc is an infix operator whose meaning depends upon the types of its operands. In this example DATA is a data item. K is assumed to be a key enciphered by the master key and is deciphered. It reveals itself as a data key and is used to encipher DATA. The result is stored in the item called E_KDATA. Following Reference 1, the subscript is part of the name of the item, intended to make its contents clear to the reader, and does not indicate any operation.

2 DATA := K dec E_KDATA;

This is the inverse operation to Example 1. K is deciphered using the master

key and reveals itself as a data key. It is used to decipher the item E_KDATA and the result is stored in DATA.

3 $E_{KK}K$:= KK enc K;

This is at first sight the same operation as Example 1. KK is deciphered using the master key. It reveals itself however to be a KEK. Therefore the second parameter of *enc* is deciphered using the master key in order to provide a parameter of type 'key'. This having been done it is enciphered using the clear form of KK. The result of this operation is stored in $E_{KK}K$. The names of the items are chosen to indicate their contents and have no typing significance to the mechanism, which operates according to the revealed types. This is the operation of 're-encipher from master key'.

4 K := KK dec $E_{KK}K$;

This is the inverse of Example 3. The value in $E_{KK}K$ is deciphered. The key needed to do this is obtained by deciphering the value of KK using the master key. The revealed type of $E_{KK}K$ must accord with the type of the key obtained from KK, in this case assumed to be a KEK. Before being stored in K the key obtained by 'KK dec $E_{KK}K$' is enciphered using the master key. This is the operation of 're-encipher to master key'.

5 $E_{KK1}K$:= KK1 enc (KK dec $E_{KK}K$):

Assuming that the names of items are chosen to indicate the contents and therefore the required types, this example causes a key enciphered by KK to be deciphered and then re-enciphered using a different key-enciphering key, KK1.

6 DATA := KKK dec E_{KKK}KK dec E_{KK}K dec E_KDATA;

This example shows the use of hierarchy of keys which must be successively deciphered to finally decipher the value in E_KDATA and store it in DATA. Notice that the order of operations is assumed to be left to right. An operation such as

K1 dec (K2 dec E_{twice}DATA)

is possible and makes sense if the value in E_{twice}DATA has been obtained by enciphering the clear value first by K1 and then by K2.

6 Public keys

Referring once more to Fig. 8, two bits of the extra 8 bits accompanying the key have been assigned to indicate whether the key may be used for enciphering and deciphering. Since the key and its type information are securely bound when the item containing the two is enciphered the same key typed in one case as 'encipherment only' and in the other case a 'decipherment only' gives us a kind of public key system, provided that no-one, apart perhaps from some ultimate trusted authority,

either knows master keys or can discover any values held in clear in the secure modules. This is, perhaps, a generous assumption in a system which requires very high security but it may well be commercially useful.

A particular use of 'encipherment only' type is to use an 'encipherment only' KEK at one installation to encipher a key to be used at another installation. The receiving installation holds the equivalent key deciphering key but the sending installation does not. If a spy with access to the sending installation intercepts the transmission he cannot decipher the key. This problem is solved in Reference 1 by a second variant of the master key.

7 Other uses of typed keys

Typed keys may be useful in the protection of software copyright. A conceptually simple scheme is to provide a secure instruction store and to use typed keys to provide it with copyright programs. Such a program would be enciphered by the owner of the copyright. The key used to encipher the program would be typed as 'obey only', using another of the spare 8 bits. The enciphered program would be sent to all authorised users enciphered by the single common 'obey only' key. The 'obey only' key would then be enciphered by an individual KEK for each authorised user. This KEK would be made available to the secure module of the individual user but would not be known to the user himself. It could be his installation master key. For greater flexibility and to enable the same version of the 'obey only' key to be used at several installations it could be a KEK enciphered by the master key. In an emergency this KEK could be sent to a designated standby installation and installed there, enciphered by that installation's master key. The copyright software could then be used securely at the second installation.

A 'secure instruction store' of course implies that the secure module has grown to encompass rather more of the computer installation; there is scope for refining the principle to protect just enough of the copyright program to make stealing sufficiently improbable.

A similar technique can be used to ensure that amendments to secure software are only made by the copyright owners but avoiding the need to re-issue the whole program. The enciphered patch would require decipherment by a key which was tagged to indicate to the logic in the secure module that it was an authorised patch to an identified program to be used only for that purpose. This key would itself be enciphered for each user individually in the same way as the 'obey only' key.

Reference 8 introduces a technique to prevent authorised users of the encipherment facility from misusing each others data keys. It does this by using the authorised user's identity, as well as a KEK kept securely within the encipherment facility, as parameters to the function which enciphers a data key. A user must identify himself by submitting a password to gain access to the facility and his identification is used when deciphering any key on his behalf. The effect is as if each user were forced to use his own version of the master key. Typed keys, as described in this paper, may be used to give the same protection. If a user must identify himself to the encipherment module, or the operating system does it on

his behalf, then type information revealed when a key is deciphered inside the module may be used to invoke logic which involves the user's identity in further use of that key.

The principle of using a key's type to restrict the use of the data it enciphers may be applied more generally. One might, for example, have logic built into the secure module which produced statistics. The type information of the key would contain an identification of the procedure, built into the secure module, which must be applied to the deciphered data. It would thus be possible to send a user sensitive information which he must not see but from which he could produce statistics.

8 The Mechanism of the encipherment/decipherment module

From the examples given when discussing the proposed notation it can be seen that the secure module must contain a stack for its operands. The notation is interpreted making use of the more primitive operations described in Section 4, which themselves use the stack. An item on the stack is tagged as an address, a word or a key (DEK or KEK). The master key is held off stack in the secure module. Apart from the master key, clear keys are held only on the stack.

An interpretation of example 5 from Section 5 is shown below. Assume that the interpreter is presented with Example 5 as 10 separate items i.e.

$E_{KK1}K$:= KK1 enc (KK dec $E_{KK}K$) ;

It proceeds as follows. The stack grows downwards.

(i) Read '$E_{KK1}K$'. Place its address on the stack.
stack $E_{KK1}K$ address, *tag* address

(ii) Read ':='. Keep it until both its operands are available.

(iii) Read 'KK1'. Place its address on the stack.
stack $E_{KK1}K$ address, *tag* address
KK1 address, *tag* address

(iv) Read 'enc'. Keep it until both its operands are available.

(v) Read '('. Keep it until its matching')' is available.

(vi) Read 'KK'. Place its address on the stack.
stack $E_{KK1}K$ address, *tag* address
KK1 address, *tag* address
KK address, *tag* address

(vii) Read 'dec'. Keep it until both its operands are available.

(viii) Read $E_{KK}K$. Place its address on the stack.

stack $E_{KK1}K$, *tag* address
KK1 address, *tag* address
KK address, *tag* address
$E_{KK}K$ address, *tag* address

(ix) Read ')'. 'dec' may now be evaluated. It takes the top two items from the stack. KK's address is used to obtain its value. This is necessarily a word. It is therefore deciphered by KM to produce a key, the tag showing it to be a KEK (in our example) being obtained from the deciphered item. $E_{KK}K$'s address is used to obtain its value. This is also necessarily a word. The KEK now available as the first operand is used to decipher it. The value thus revealed is placed on the stack, tagged as a key. Additional type information in the tag is obtained from the deciphered item.

stack $E_{KK1}K$, *tag* address
KK1 address, *tag* address
K, *tag* DEK (or KEK would suit the operations shown here)

The ')' may now be dealt with to ensure that all operators met since the bracketed expression started are evaluated and the result left on the stack. The effect, in this case, is to leave the stack unchanged.

(x) Read the ';'. Since it shows the end of the statement the second operand of 'enc' has been found and "enc" can be evaluated. It takes two items from the stack. It uses the first as its key, first obtaining the value from the address given. This is necessarily tagged as a word, being the external form of KK1. KK1 is therefore deciphered by KM, to reveal a key. In our example the type information thus revealed shows it to be a KEK and it is used to encipher K and K's accompanying type information. This enciphered item is placed on the stack.

stack $E_{KK1}K$, tag address
encipherment of K and its type, *tag* word

(xi) The ':=' can now be evaluated. It takes two items from the stack. The first is an address, as it requires. The second is a word. It may store this in the location indicated by the first with no preliminary manipulation. The interpretation is now complete.

9 Key generation

Some of the techniques described above depend upon the master key being unknown and inaccessible to all users of the installation. One way to achieve this is to cause it to be installed in clear by an independent authority trusted by all users. Alternatively it would be possible to supply it in a standard form and provide functions which enabled it to be changed. The user of these functions must identify his authority to change the master key and must give the values to be used by the secure module to generate a new master key. For example we might have functions

PREPARE-MASTER-KEY (OLD-SEED-1, NEW-SEED-1)
CHANGE-MASTER-KEY (OLD-SEED-2, NEW-SEED-2)

The assumption is that the secure module, when it is first used, contains three values which we may call the master key, seed 1 and seed 2. They are standard published values. When the function PREPARE-MASTER-KEY is used the value of its parameter OLD-SEED-1 must be the same as seed 1. This establishes the authority to use the function. The first time this is done it is, of course, valueless as authentication. If the check is successful the value of the parameter NEW-SEED-1 replaces the stored seed 1, the master key is deleted and the secure module enters a state in which the only function it will accept is CHANGE-MASTER-KEY. When CHANGE-MASTER-KEY is used there is, similarly, a check that OLD-SEED-2 is the same as seed 2. If it is, NEW-SEED-2 is stored as seed 2 and a new master key is generated, using seed 1 and seed 2 and using the basic enciphering operation (e.g. DES) to generate a pseudo random number. It is important that the process be repeatable so that the master key may be created again outside the secure module in case of accident.

Once one has a master key and the ability to change it in a controlled fashion the potential is available to create data keys and key enciphering keys for use in the installation. Functions to do this may be supplied to authorised users and may make use of the master key and the basic enciphering algorithm. Such keys would never appear outside the secure module, except in enciphered form. This topic is discussed, for example, in Reference 4. It is desirable that a different procedure be used by the secure module to create KEKs and DEKs so that a user cannot deliberately create one of each kind where the clear form is identical. A suitable difference for example might involve enciphering using the master key, adding a constant dependent on the type of key being created and enciphering again as part of the procedure. This difference in creating key values must not, of course, apply to enciphering and deciphering versions of the same key. The function available to create a key should have a parameter which specifies the type information to be recorded with the key.

The users must be provided also with a means of transmitting keys in a secure way to other installations. This is discussed, for example, in Reference 1 and Reference 6. The intention here is not to contribute new ideas on the topic but to comment on how it would be affected by the use of typed keys. The essential point is that a key-enciphering key must be available both at the location whose users wish to communicate and at some other location. The other location may be a key distribution centre or the intended correspondent for enciphered messages. The KEK may be the master key of one of the installations or a specially designated transmission key. Let us assume for the present discussion that 'genuine' public keys are not available (i.e. those of Reference 7 where the clear version of an enciphering key is no help in discovering the corresponding deciphering key). Now the methods described here differ from those of ref. 1 in that the clear form of the master key of a particular installation must be unknown even to the most privileged user of that installation if best use is to be made of the facilities. A possible means of supplying the transmission key is to allow a privileged independent authority to input it in clear at the two installations which are to communicate. In this case, however, it must not be possible to obtain the master key enciphered by the transmission key

since, if it were, the privileged authority could obtain the clear form of the master key and the security given by PREPARE_MASTER_KEY and CHANGE_MASTER_KEY would be rendered useless. A better solution is to provide two functions similar to those suggested for creating a master key. The transmission key would be installed at each of the two installations which are to communicate by the use of the two functions, each by an independent user.

10 Effect on data encipherment standards

Reference 2 states that the extra 8 bits which accompany the key used by the DES algorithm are used for error detection. The draft international specification for DEA-1, which is based on Reference 2, has not defined the use to be made of these 8 bits; their use as described here is therefore allowed. This paper has sought to show that they may usefully be used to show the key's type and that an international standard deriving from Reference 2 should not insist that they be used to record parity. Standardisation of their use to record the kind of information described in this paper may prove useful.

11 Conclusions

If the information describing a key's intended use is stored with the key and the resulting item enciphered so that it can only appear in clear inside the module which performs encipherment and decipherment, then greater control can be exercised on the use of keys. This control can be put to good use, both in manipulating keys and in restricting them to a particular use. Moreover the technique accords well with the use of a clear notation to describe key manipulation.

Acknowledgment

The author would like to acknowledge helpful discussion of the ideas in this paper with Mr D.W. Davies and Dr. W.L. Price of The National Physical Laboratory, Teddington.

References

1 EHRSAM, W.F., MATYAS, S.M., MEYER, C.D. and TUCHMAN, W.L.: 'A cryptographic key management scheme for implementing the data encryption standard', *IBM Systems J.*, 17, (2).

2 'Data encryption standard'. Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, US Department of Commerce.

3 'DES modes of operation'. FIPS Publication 81, National Bureau of Standards, US Department of Commerce.

4 MATYAS, S.M. and MEYER, C.H.: 'Generation, distribution and installation of cryptographic keys', *IBM Systems J.*, 17, (2).

5 PARKER, T.A.: 'Security in a large general-purpose operating system: ICL's approach in VME/2900'. *ICL Tech. J.*, 1982, 3(1) pp. 29-42.

6 PRICE, W.L. and DAVIES, D.W.: 'Issues in the design of a key distribution centre'. NPL Report DNACS 43/81, National Physical Laboratory, Teddington, Middlesex UK.

7 RIVEST, R.L., SHAMIR, A and ADDLEMAN, L.: 'A method of obtaining digital signatures and public key cryptosystems'. Communications of the ACM, 2nd February, 1978.

8 SMID, M.E.: 'A key notarization system for computer networks'. NBS Special Publication 500-54, National Bureau of Standards, US Department of Commerce.

The use of COBOL for scientific data processing

Tim Harle and Richard Carpenter
Hydrographic Department, Ministry of Defence (Navy), Taunton, Somerset

Abstract

This paper describes various advantages found by a production team in the use of COBOL for certain tasks when processing scientific data. The historical background leading to the introduction of COBOL is outlined, as are the main features of the language. After considering the mechanisms of mixed-language programming, the facilities of COBOL are compared with those of the more usual scientific languages, with particular reference to FORTRAN. The main advantages of COBOL are seen to be in efficiency of processing, file and data handling, widespread availability and manufacturers' support, including provision for the future.

1 Introduction

This paper aims to show how a language not conceived for scientists can be used successfully for processing scientific data. As such, it does not call on scientists to make a headlong rush into writing programs in COBOL, but rather seeks to provide a stimulus to further thought that what at first sight seems unlikely can be of great benefit. It represents the fruits of a production team having to make the best use of the tools available, rather than a research one; it is therefore essentially constrained to a discussion of COBOL and FORTRAN, although other languages such as Pascal and ALGOL-68 are not ignored. The paper makes no claims about the elegance of either COBOL or FORTRAN: both were developed some time before the principles of program design were properly understood.[1]

The paper is illustrated at various points by real life examples, so it may help to set the scene by explaining briefly what the Hydrographic Department is and how scientists came to start using COBOL.

2 The development of computing at the Hydrographic Department

The chief role of the Hydrographic Department is the supply and continual updating of a world series of charts and other marine publications for the safety of both the Royal Navy and the mariners of Britain and many foreign countries. The Department was founded in 1795: since that date it has built a worldwide reputa-

Note: The views expressed in this paper are those of the authors and not necessarily those of the Hydrographic Department.

tion for reliability. One of the prime reasons for this is that Admiralty charts are kept updated manually to the point of sale, for instance with newly discovered shoals or recent wrecks, and there is a free service whereby weekly Notices to Mariners give details of all changes to be made to published charts, not to mention a round-the-clock radio service to warn about up-to-the-minute navigational dangers. This reputation has meant that some 70% of the Department's annual turnover of £10M is in exports, a fact recognised in 1978 by the award of a Queen's Award for Exports – a highly unusual achievement for a part of the Ministry of Defence.

Computers began to be used in the Department in the late 1950s, particularly for the complicated mathematics involved with geodetic calculations. A good example is provided by the radio position-fixing systems such as Decca Navigator, which rely on hyperbolic interference patterns to obtain a position. The equations involved with translating these hyperbolae from a spheroidal earth were somewhat complex; nevertheless, they reduced to a set of formulae which could be converted into computer programs – first FORTRAN II and then ALGOL (60 as it would now be called) were used for this type of work.

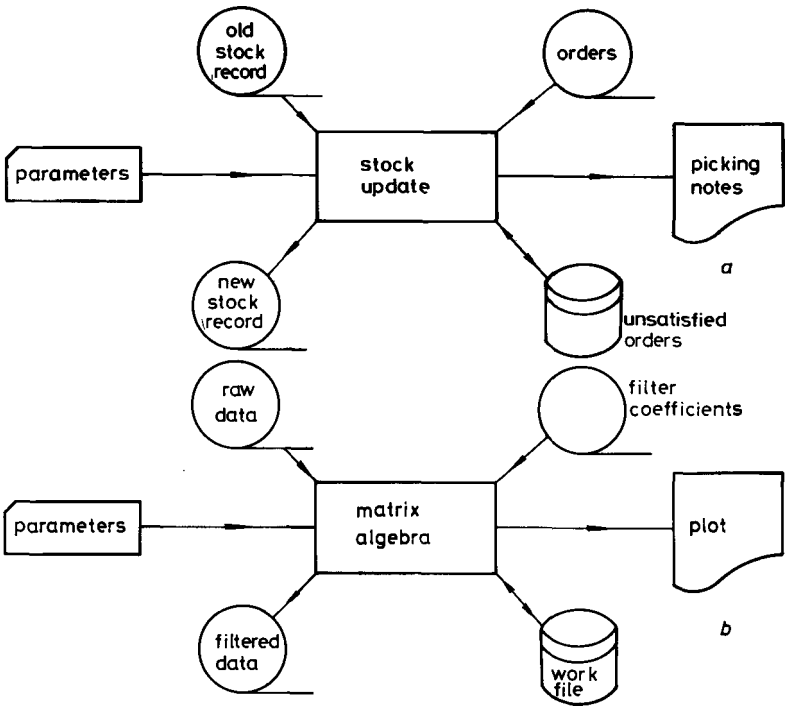


Fig. 1 Computer run charts
(*a*) Stock control
(*b*) Inertial navigation

Such mathematical applications, which gave a reduction of up to 40:1 in the time taken to draw an overprint for a Decca chart, formed the backbone of the work. This enabled other applications, many of them with graphics, in such fields as surveying, oceanography, tides and geophysics to be automated. Such tasks led to

ICL 1900 mainframe equipment being supplemented by a variety of machines including an ME29 programmer workbench, photohead flatbed plotters, interactive graphics editing systems and speech recognition equipment.

In the meantime, elsewhere in the Department, there were 4000 publications, 2500000 shelf items and a large printing works feeding a worldwide distribution network with 200 agents – a straightforward stock control and accounting application for automation using the language that had become widespread in the business world: COBOL.

So two programming teams were working in different languages under different management structures until a manpower review team in 1975 suggested that they should amalgamate. Once this happened, it was soon realised that there were similarities between the different disciplines, often reflected at a system level by similar-looking computer run charts (Fig. 1). This combining of effort coincided with a time when some of the files of scientific data, particularly in the oceanographic and surveying fields, were getting so large that inefficiencies in their processing were beginning to have a marked effect on the mainframe's throughput. And that is how the Department came to start experimenting with COBOL for scientific work.

3 Main features of COBOL

COBOL stands for Common Business Oriented Language. The US Navy, Capt. Grace Hopper and 1959 are usually acknowledged as parent, midwife and date of birth, respectively. The first ANSI definition was published in 1968,[2] followed by a second in 1974[3] and a third is now due for publication in early 1983. In the light of the debate over the core-plus-modules approach to the next FORTRAN standard, it is interesting to note that these COBOL definitions are in modular form.

```
ADD SUB-TOTAL TO RUNNING-TOTAL
MOVE 9999 TO DEPTH
DIVIDE HIT-COUNT BY 100 GIVING HIT-PERCENT ROUNDED
INSPECT SORT-KEY REPLACING LEADING SPACES BY ZEROS
PERFORM INPUT-ROUTINE UNTIL DUE-DATE EQUALS TODAY
```

Fig. 2 Typical COBOL commands

COBOL syntax is based on a series of words giving a finished product closer to plain English than the more mathematically based languages such as ALGOL and FORTRAN (Fig. 2). This is both a strength and a weakness: it can help considerably in writing easy to understand, and therefore to maintain, programs, or it can produce a verbose mess.

The language was one of the first to recognise the important distinction between data and the actions performed on those data. This gives rise to the two major divisions of a COBOL program: the DATA DIVISION and the PROCEDURE DIVISION. A program is completed by the IDENTIFICATION DIVISION and the ENVIRONMENT DIVISION. It is this distinction between data and procedures

which allows COBOL to be so useful for processing scientific data: with many of the Department's systems, the data are scientific, e.g. oceanographic temperatures, salinities, conductivities and pressures, but the applications involve such conventional tasks as sorting, listing, updating and selection.

Before considering the DATA DIVISION, mention should be made of the ENVIRONMENT DIVISION where, among other things, files are assigned to hardware devices. It is in its file handling that COBOL often proves most useful: a file's ORGANISATION may be either SEQUENTIAL or INDEXED whereas its ACCESS MODE may be SEQUENTIAL or RANDOM. Typical applications are in areas where data are held on a grid: when the grid gets too big to be held in main store, an indexed sequential file can take over. Although the introduction of virtual machine architecture has eased this constraint, the problem remains. Examples in the Department's sphere of work are provided by any parameter that has to be considered over the surface of the earth, e.g. gravity or geomagnetic fields. Holding data on a 1° grid requires 64 800 items; at $1'$ intervals this expands to $2{\cdot}3 \times 10^7$. And it is by no means unusual to require more than one value at each point to define such fields completely. In such cases, the processing programs are in FORTRAN but the file access software is in COBOL.

The DATA DIVISION is split into sections, of which two can be considered: the FILE SECTION and the WORKING-STORAGE SECTION. Within the FILE SECTION, all records for input and output are defined. A particular record, or part of a record, can be defined in more than one way using the REDEFINES clause; thus a program need not know in advance what the next record to be read will contain. This can be of assistance in the processing of data collected from a variety of sensors in real time; for example, the Navy's four largest survey ships have real-time systems onboard handling interrupts from several devices – depth recorders, gravimeters, magnetometers and navigation aids, including satellites – so the files of data they produce contain different record types in an unpredictable order.

Variable length records can be handled with ease by use of the OCCURS DEPENDING clause: this has a particular application where a sensor may function for an unspecified length of time. An example of this is in the field of oceanography where data are captured by releasing instruments over the side of a ship that continue recording until they reach the bottom. The Pacific ocean contains trenches up to 10km deep, so recording to this depth must be catered for, but the great majority of observations are well under 1000m, many under 100m. Using fixed length records, as required by standard FORTRAN, the master file extended over 12 reels: with variable length records this reduced to three. ALGOL, with its stream- rather than record-oriented input/output, is also less constricting than FORTRAN.

In the DATA DIVISION as a whole, all data items to be used must be declared (thus avoiding FORTRAN's weakness that caused the Mariner spacecraft to miss Venus[4]). Typing is stronger than in FORTRAN and ALGOL, but not as strong as in later languages such as Pascal; such symbols as A, 9 and X are used for letters, numbers and alphanumerics, respectively, and each field is given a width, implied decimal point if required and an optional initial VALUE and range. Individual items may be considered together as group fields with a hierarchy of levels up to a full

record: a good example is provided by geographical positions (Fig. 3). This greatly simplifies data handling compared with FORTRAN which recognises no data item except the individual variable or array. COBOL also uses arrays or tables; like FORTRAN and Pascal, but unlike ALGOL-68, array arithmetic cannot be performed.

```
01      GEOG-POSITION.
    02      LATITUDE.
        03      LAT-DEG         PIC 99.
        03      LAT-MIN         PIC 99V999.
        03      LAT-HEM         PIC A.
    02      LONGITUDE.
        03      LON-DEG         PIC 999.
        03      LON-MIN         PIC 99V999.
        03      LON-HEM         PIC A.
```

(*a*) DATA DIVISION

```
SUBTRACT 1 FROM LON-DEG
MOVE LATITUDE TO RECORD-INDEX
WRITE GEOG-POSITION.
```

(*b*) PROCEDURE DIVISION

Fig. 3 Use of group fields

Individual fields may be in either character (DISPLAY) or binary (COMPUTATIONAL, abbreviated COMP) format. Different types may be included within the same record; such mixing is not allowed in standard FORTRAN where whole files must be of one type or the other; Pascal even uses different commands for the different types of file. The mixing of types is illustrated by a system handling wreck information in UK waters: the positions of wrecks, upon which mathematics has to be performed for extraction and plotting, are held as binary fields whereas the ships' names and other textual details are in character form. Floating-point fields are not supported, although some compilers, such as C2 on VME 2900s, have non-standard extensions to allow this.

The actions supported in COBOL's PROCEDURE DIVISION were obviously thought out some years before Dijkstra's celebrated paper[5] or Jackson's well-known methodology[6] on what has become known to us as structured programming. Nevertheless, the three basic building blocks are there (Fig. 4): the sequence, brought together as a series of verbs in a paragraph or block; the selection, IF...ELSE..., supported at last in FORTRAN 77[7]; and the iteration, PERFORM...VARYING...UNTIL... giving almost the full DOWHILE construct. It is thus possible to write well structured programs in COBOL, even if the elegance of the block-structuring of ALGOL and its derivatives is missing.

Many other procedures are available, though not necessary widely used, including the notorious GO TO. Subroutines are supported in COBOL with transfer of data

```
    PERFORM START-PARA.
START-PARA.
    ACCEPT TODAY FROM DATE
    OPEN OUTPUT PRINT-FILE
    WRITE HEADING AFTER PAGE-THROW.
```

(*a*) Sequence

```
IF DEPTH-IN-RANGE
    ADD 1 TO GOOD-COUNT
    MOVE REF-NO TO DEPTH-TABLE (GOOD-COUNT)
ELSE
    ADD 1 TO BAD-COUNT
    PERFORM ERROR-ROUTINE.
```

(*b*) Selection

```
PERFORM CHECK-ROUTINE VARYING
        CHART-NO FROM 1 BY RATE UNTIL
        DISCREPANCY GREATER THAN 50.
```

(*c*) Interation

Fig. 4 Structured programming elements

by parameter list in the LINKAGE SECTION. The inline SORT can simplify programming, but care should be taken over its use because of operational inefficiencies compared with free-standing sorts. 'Casual' input/output is allowed with the ACCEPT and DISPLAY commands: FORTRAN 77, ALGOL-68 and Pascal all have essentially similar instructions, although COBOL's verbs are now forming the basis of communication modules in TP applications. They can also be of use during software development, as can DEBUG lines whereby lines with 'D' in column 7 are included only for testing. The inline COPY facility can also speed up program development by allowing whole sections of common code to be copied from libraries, e.g. file descriptions; however, its use has probably declined with the move away from batch development and the introduction of powerful screen editors.

The above has outlined some of the main features of ANSI COBOL, commonly found on both mainframes and minis; the language's widespread availability, and hence potential portability, extends to micros where Microfocus' CIS (Compact Interactive Standard) COBOL runs under both CP/M and UNIX operating systems.

4 Mechanism of mixing languages

There are two main ways in which languages can be mixed: at a system level, individual programs can be written in, for example, COBOL and FORTRAN, linked by transfer files; within a program, individual subroutines can be in different languages. The former requires common file protocols between different compilers, the latter requires a common binary symbolic format (semicompiled in ICL's 1900

terminology, OMF for 2900s) for link-loading (consolidation and collection for 1900 and 2900, respectively). The program calls are straightforward (Fig. 5).

```
CALL ALGOL 'VECTOR' USING X Y
CALL FORTRAN 'SQUARE' USING SIDE UNIT-AREA
```

(*a*) Calling from COBOL

```
CALL READER (NOREAD, NOHITS)
```

(*b*) Calling COBOL from FORTRAN

Note: Calling non-ALGOL-68 procedures from ALGOL-68 is not strictly allowed, but the RS family does allow this.

Fig. 5 Linking by subroutine

Both methods may be illustrated from the system mentioned above that handles information on some 20 000 wrecks around the British Isles. The two principal outputs are printouts with textual details about particular wrecks and graphics showing wrecks plotted on a particular projection and scale. Apart from these functions, the system includes normal updating, validation, etc. The majority of these tasks is performed in COBOL with two exceptions: a FORTRAN library subroutine is available to tell if a point is within a given polygon and the plotting is performed by a FORTRAN program.

There are weaknesses in both mixing methods. Having separate programs can cause overheads in both software development and execution. Subroutines have restrictions on data types: in general, integers and characters cause no problems, other data types and arrays require manipulation. However, tests have shown that the overhead of splitting real numbers into a series of integers can be more than outweighed by the reduction in execution times (see below).

5 Relative strengths of COBOL and FORTRAN

The description of COBOL above was set in the context of other languages; it may be instructive now to make some specific comparisons with FORTRAN. It must be admitted at the outset that the Department's initial reason for experimenting with COBOL was on the grounds of operating efficiency – a less fashionable idea in these days of cheap hardware. This particularly applied in areas with high volumes of data, for example in the shipborne surveying systems mentioned previously.

These record a 2kb record every minute and can survey more or less continuously 24 hours a day for three weeks at a time. With some inertial navigation processing requiring retrospective solution of five simultaneous equations by matrix algebra every 10 seconds, there was a problem. Use of FORTRAN to simplify the maths would introduce long run times because of the time taken to process the magnetic tape files; use of COBOL to get the data in and out fast would make it almost impossible to perform the inertial calculations.

two points are worth making: first, given the enormous investment in COBOL, it will be many years hence before its use wanes and, secondly, even if COBOL is superseded, whether by Ada or some other language(s), it is unlikely that the programming community will be left without aids to the conversion. There has been a glimpse of this with the source convertor provided by ICL for Range COBOL to assist in the translation of ANSI68-based programs to the ANSI74 standard:[18] this project, code named C2*, was at one stage put in abeyance but was restarted, significantly, after pressure from users.

7 Conclusion

This paper has sought to show the benefits of COBOL to one scientific group of users. If it has a message, it is that products should be judged on their uses rather than their reputation.

To take up a metaphor quoted some years ago by Professor David Barron, it may appear that switching from FORTRAN to COBOL is akin to evolving from one doomed species of dinosaur to another.[19] But it should not be forgotten that the only reason the various types of dinosaur lasted as long as they did (some 165 million years) was because they were so well fitted for the conditions around at the time.[20] And when the evolutionary process takes its course and the time comes to add this dinosaur to the Natural History Museum's collection, there remains the hope that it will be a co-operative beast rather than a predator that takes over.

References

1 SHAW, M., et. al.: 'A comparison of programming languages for software engineering', *Software – Practice and Experience*'. 1981, **11**, 1-52.
2 American National Standard COBOL, X3. 23-1968, ANSI, 1968.
3 American National Standard COBOL, X3.23-1974, ANSI, 1974.
4 HOARE, C.A.R.: 'The emperor's old clothes', *Comm. ACM.*, 1982, **24**, 75-83.
5 DIJKSTRA, E.W.: 'Programming considered as a human activity', 1965 IFIP Congress
6 JACKSON, M.: *Principles of program design,* Academic Press, 1975.
7 American National Standard FORTRAN, X3.9-1978, ANSI, 1978.
8 COBOL, 3rd edn., TP4427, ICL, 1976.
9 Extended FORTRAN, TP4269, ICL, 1971.
10 Range COBOL Language, RP1107, ICL, 1980,
11 CALDERBANK, V.J. and PRIOR, W.A.J.: 'The GHOST graphical output system users' manual', CLM-R 177, UKAEA, 1978.
12 AL-JARRAH, M.M and TORSUN, I.S.: 'An empirical analysis of COBOL programs', *Software - Practice and Experience*, 1979, **9**, 341-359
13 VAN WIJNGOORDEN, A. et al.: 'Revised Report on the Algorithmic Language ALGOL-68', *Acta Informatica*, 1977, **5**, 1-236.
14 Pascal, BS6192, BSI, 1982.
15 PL/1(F) Language Reference Manual, GC29-6515-8, IBM, 1971.
16 American National Standard PL/1, X3.53-1976, ANSI, 1976.
17 BARNES, J.: 'An Overview of Ada', *Software - Practice and Experience*, 1980, **10**, 851-887.
18 Converting COBOL Programs to Range Standards, TP3760, ICL, 1980.
19 BARRON, D.: 'Alternatives to the dinosaur: the case against FORTRAN', *Computing*, 24 November, 1977.
20 COLBERT, E.H.: 'Giant Dinosaurs', *Trans. New York Academy of Sciences*, 1955, **17**, 199-209, reprinted in Preston Cloud (Ed.), *Adventures in Earth History*, 716-723, W.H. Freeman & Co., 1970.

Recognition of hand-written characters using the DAP

JOSEF PECHT *
Lehrstuhl C Fuer Informatik der TU Braunschweig, Gausstrasse 11
D-3300 Braunschweig, West Germany
and
ISA RAMM
Im Gettelhagen 98, D-3300 Braunschweig, West Germany

Abstract

A new method for recognising hand-written characters, which does not extract a skeleton of the input character, has been programmed for the DAP. In this procedure, the hand-written input pattern to be investigated undergoes different degrees of 'thickening'. One assumes here that each letter of the alphabet is represented by several typical forms. Each of these forms consists of a chain of line segments. To decide what character is represented by a given input pattern, the number of these forms that occur in the 'thickened' character are counted and compared. The character is then thickened and the count repeated. The form with the highest number of matches identifies the input character. Because of the high computational effort required, this procedure is not suitable for serial computers but is very suitable for bit-plane computers with parallel processing such as the DAP.

1 Introduction

To be able to recognise hand-written characters one proceeds in general from the following statement of the problem:

> A processor or a program that analyses a digitised photographic representation of a hand-written character and takes a decision as to which ideal character it could be needs to be constructed. The design of such a system is based on the idea that the idealised characters can be described by polygonal linear structures, that the writer has attempted to imitate these linear structures and that the resulting written character may be distorted by uncontrolled movements and characteristics of the writing implement, as well as by extraneous noise due to the transmission process.

A number of attempts to solve this problem both in theory and practice have been made. The process is frequently divided into four steps [1]. In the first phase, called 'smoothing', any transmission errors and unimportant local effects such as noise, small interruptions and shaking movements are eliminated. This results in a rep-

*Now with: Institut fur Material und Festkorperforschung III, Kernforschungs-Zentrum Karlsruhe, Postfach 3640, D-7500 Karlsruhe, FRG

resentation that consists of a series of lines of varying thicknesses but with clean edges. Based on the idea that the ideal characters consist of (infinitely) thin lines, a skeletisation process then follows, in which the lines of differing thicknesses change into a series of fine equally thin lines. These smoothing skeletisation phases are to be found in almost all procedures for the recognition of hand-written characters.

As far as the next steps are concerned, one can divide the usual procedure into two different types which are basically different from each other. In the first one, a line tracing routine is put onto the skeleton and a structural description of the line structure is produced. This phase of this structure extraction, which essentially produces a system of relationships between parts of lines and cross-over points, is followed by a second phase in which the structure recognised is compared with some ideal model structures (letters of the alphabet) and a decision is taken.

The second type of algorithm attempts, without describing the structure, to extract certain characteristic vectors from the skeleton image, such as the number of intersections made by certain lines, the sum of points in rows and/or columns etc. The actual classification is done by the usual methods of statistical pattern recognition[2].

The procedures just described were developed primarily for traditional serial computers. Their notable characteristic is the change in the data structures used during processing. During the smoothing and skeletisation phase, one operates on two-dimensional digitised perhaps black and white (0-1) patterns, while in both the last steps lists or n -dimensional real vectors are manipulated.

The question now arises whether it is sensible to use these methods without further investigation on parallel, particularly bit-plane, computers such as the DAP. The answer is not completely unequivocal. It seems that 'thickening' and skeletisation can be carried out very quickly on parallel computers. However, using a suitable image storage technique, these operations can be considerably speeded-up using serial computers. [1] Also, these first two steps need relatively little computing time compared with the last two. The questions as to whether list structures can be advantageously processed on parallel computers is still open. The statistical decision program can be much more advantageously parallelised, something which cannot necessarily be said for the procedure for extracting patterns from the skeleton. We cannot therefore assume that these procedures can be converted as they are to run on bit-plane parallel computers, although it is worth investigating this further in special cases. These considerations, together with the point of principle, that new computer architectures call for new models and new solutions, led us to design a new method which is tailor-made for bit-plane computers and which differs fundamentally from traditional methods, being most unsuitable for serial computers.

2 Recognition of hand-written characters in bit-plane parallel computers

2.1 The basic idea, demonstrated by using a simple model alphabet

The method proposed here was originally introduced by Pecht[3] and uses the

following process: the binary input pattern m, representing the hand-written character to be identified, is thickened continuously until it becomes one solid black field.

Let m^0 (=m), m^1, m^2, ... m^i denote the successively thickened versions of m. In Fig. 1 this sequence is sketched for a particular input pattern m and a thickening template T defined as $\begin{matrix}\blacksquare\blacksquare\\ \times\blacksquare\end{matrix}$ – that is, each point of m is replaced by this matrix of six points. m here represents a hand-written character 'X', slightly adorned with flourishes.

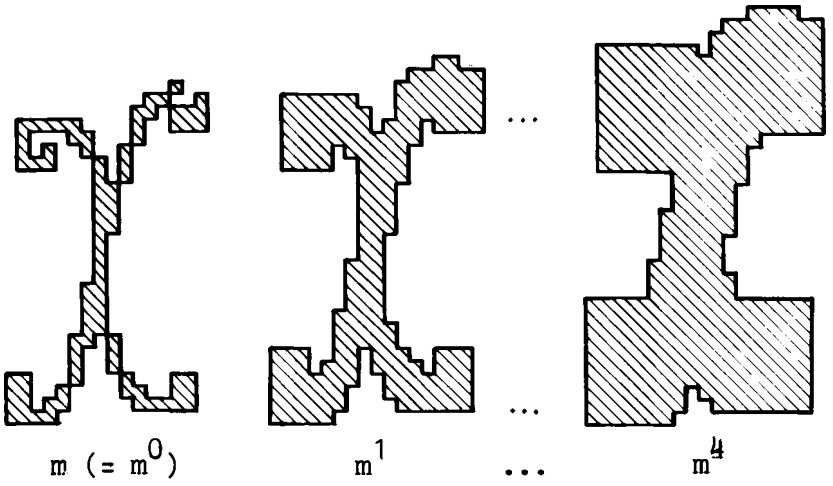


Fig. 1 Typical input pattern successively thickened using template $T = \begin{matrix}\blacksquare\blacksquare\\ \times\blacksquare\end{matrix}$

We further suppose that for each character α of the alphabet there exists a sequence of well-written versions V_α^j, where j = 1, 2, 3 ... consisting of polygonal curves which grow linearly in size with respect to the parameter j. The sequence of well written versions for the simple model alphabet (X, S, N) considered in the following is shown in Fig. 2. We suppose also that a *reference point* is defined for each character as the bottom left-hand corner of the minimal circumscribed orthogonal rectangle: This is shown in Fig. 3 for the model character X. The basic definition for our recognition process is:

Definition: Taking a lattice point or cell p of the cell space, we say that 'p sees V_α^j in m^i', or explicitly, 'p sees a (well written) character α of size j in m^i' if, taking p to coincide with the reference point of V_α^j, the version V_α^j is entirely contained within m^i.

Formally we can write this condition as: $p + V_\alpha^j \subseteq m^i$ is true and can shorten the expression to $E(p, i, j, \alpha)$. Fig. 3 shows an example of X of size 6 with m^4 from the input pattern m of Fig. 1.

Further, if $e(i, j, \alpha)$ is the set of all cells p for which $E(p, i, j, \alpha)$ is true and $n(i, j, \alpha)$ is the number of cells in this set, then $n(i, j, \alpha)$ is the number of cells which see a

V_α^j in m^i. The basic idea of the proposal is to use the numbers $n(i, j, \alpha)$ to bring about a decision as to which ideal character of the alphabet m could be.

i	j	N	S	X
0	1	0	0	0
1	1	22	16	22
	2	0	0	0
2	1	118	113	118
	2	5	0	5
	3	0	0	0
3	1	225	222	227
	2	54	44	64
	3			3
	4			2
	5			2
	6			2
	7			2
4	1	331	330	333
	2	153	150	160
	3	12	12	31
	4			12
	5			12
	6			12
	7			11
	8			3
5	1	434	433	434
	2	252	249	258
	3	86	83	104
	4	1	16	39
	5		12	36
	6		4	35
	7			29
	8			12

$n_{i,j,\alpha}$

i	j	N	S	X
6	1	533	533	535
	2	343	345	353
	3	169	175	195
	4	26	52	76
	5		36	62
	6		21	60
	7		3	50
	8			25
	9			3
7	1	634	636	640
	2	434	442	456
	3	254	280	296
	4	88	138	160
	5	1	72	108
	6		44	100
	7		14	78
	8			42
	9			42
		⋮	⋮	⋮
12	1	983	983	983
	2	809	809	809
	3	651	651	651
	4	509	509	509
	5	383	383	383
	6	275	275	275
	7	187	187	187
	8	117	117	117
	9	63	63	63
	10	25	25	25
	11	3	3	3

Table 1 The numbers $n_{i,j,\alpha}$ (α = N, S, X) for the input pattern m from figure 1

Table 1 shows the numbers $n(i, j, \alpha)$ for the input pattern m from Fig. 1 and the versions V_α^j from Fig. 2. If we call $n(i, j, \alpha)$ the number of 'hits' with the character α, then it is noteworthy that in this particular example, for any fixed (i, j), the number of hits with X is always greater than or equal to the number with S or N; that is, the number of correct hits is greater than the number of incorrect hits.

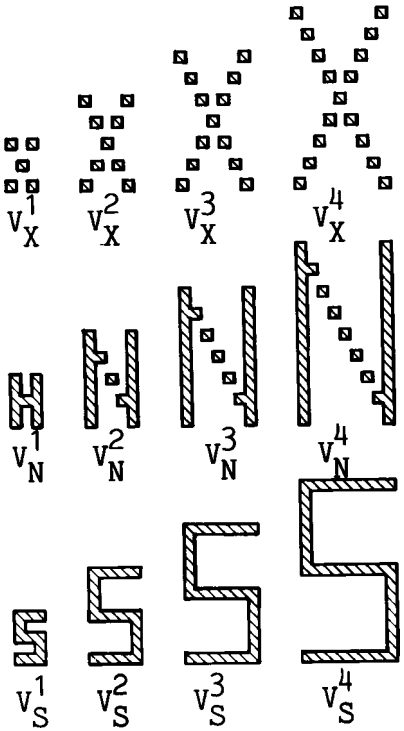


Fig. 2 Linearly growing well written versions V_α^j for the letters α = X, N and S and j = 1,2,3,4

Further, for the values i = 3,4,5 and j = 4,5,6, the number of hits with X is significantly greater than with N or S. We can now go on to ask the question, for which combinations of i and j are the number of correct hits greater than the number of incorrect hits? Fig. 4 shows some trial characters (all contained within a matrix of 16 x 32 cells, as are all the characters investigated in the course of this work) together with maps of the $(i, j,)$ space shaded to show where the majority of the hits are correct. In this example it is seen that choosing the character with the majority of hits for i = 3 and j = 4,5, or i = 4 and j = 5,6 always leads to the correct identification.

These positive results lead us to investigate the validity of the following empirical rule, which, to distinguish it from later versions, we call the 'simple majority decision rule' and abbreviate it to R.

Rule R: 1. Thicken the imput pattern m to m^i with i = 3

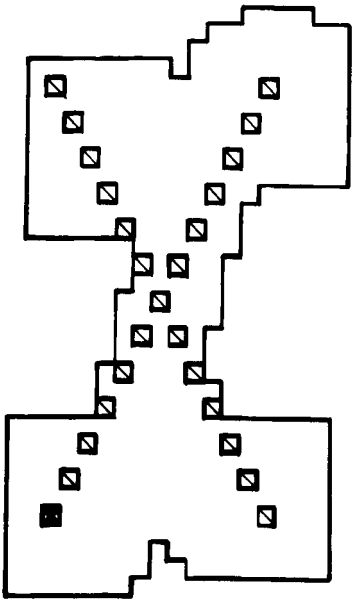


Fig. 3 'Cell x ( ) sees a well written X of size 6 in m^4 ' or, abbreviated, E (x, 4, 6, X).

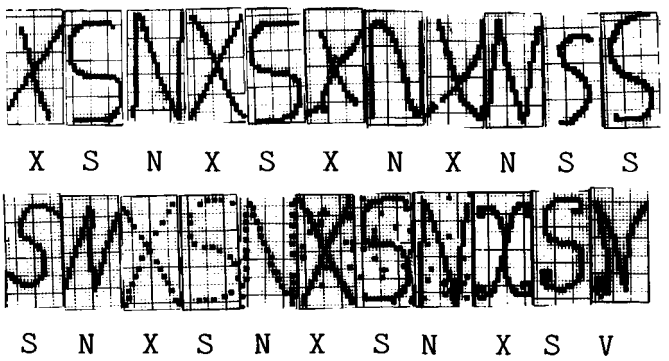


Table 2 Classification according to rule R

2. Compute $n(i, j, \alpha)$ as above for $j \geqslant 4$ and all characters α of the alphabet
3. Accept the character α for which $n(i, j, \alpha)$ is greater than for any other for the greatest possible value of j
4. If no decision results at this stage, thicken m to m^i with $i = 4$ and repeat 2,3 for $j \geqslant 5$

Table 2 shows the results of using the rule R for a series of different input patterns. The relatively high number of correct decisions shows that the idea gives a possible way ahead and that the procedure works for input characters which may slope a little, which can have flourishes on them and which are not too disturbed by

extraneous noise, provided that the person writing them keeps approximately to the pattern of polygonal curves. It must be admitted however that the trial alphabet has only a few characters and that sloping letters are not considered; we therefore modify the procedure to admit sloping versions of the alphabet as well as vertical, and we increase the number of different letters somewhat.

2.2 *Modification to the procedure*

We now assume that for each character α of the alphabet there exist a number k_α of sample (well-written) versions which we can denote by $V^j_{\alpha,k}$, where $k = 1,2 \ldots k_\alpha$.

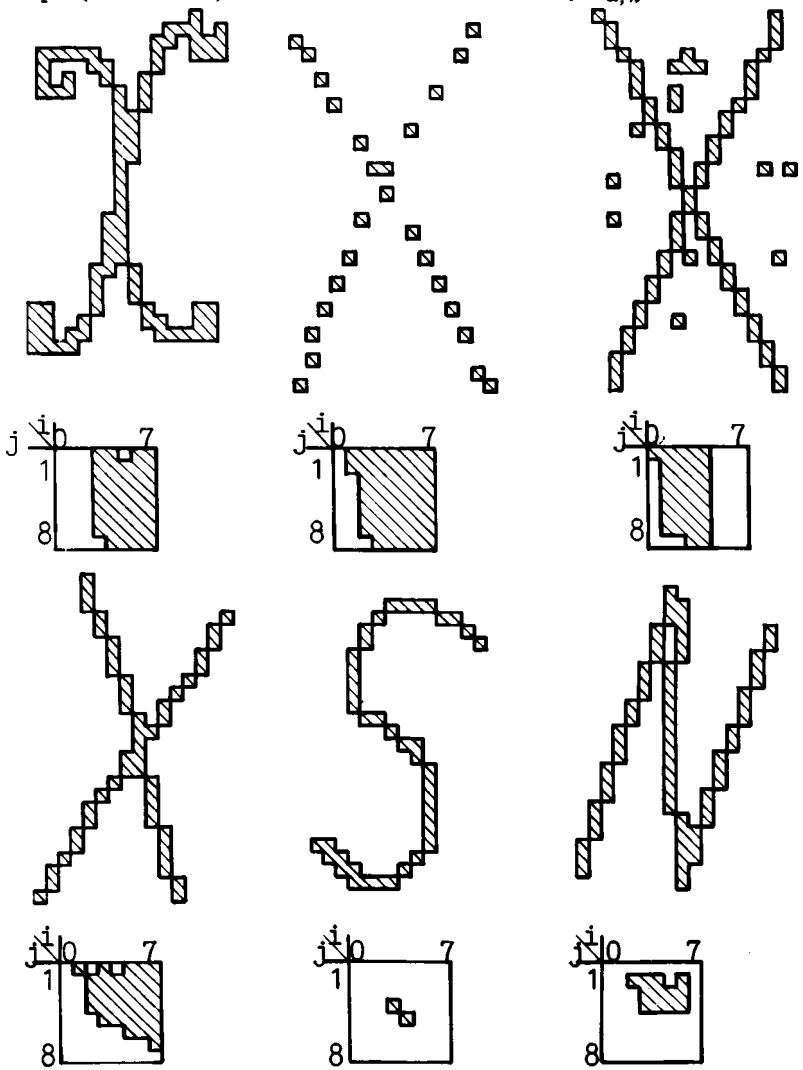


Fig. 4 Input patterns m together with the corresponding (i, j) – combinations (hashed) for which the number $n_{i,j,\alpha}$ is higher than $n_{i,j,\beta}$ where α is the correct and β represents the false characters

Fig. 5 shows three such sample versions of the latin capital A, together with their linear enlargements. We now write the expression $\hat{E}(p, i, j, \alpha, k)$ in place of the previous $E(p, i, j, \alpha)$ to represent the statement: cell p sees a well written $V^j_{\alpha,k}$ in m^i.

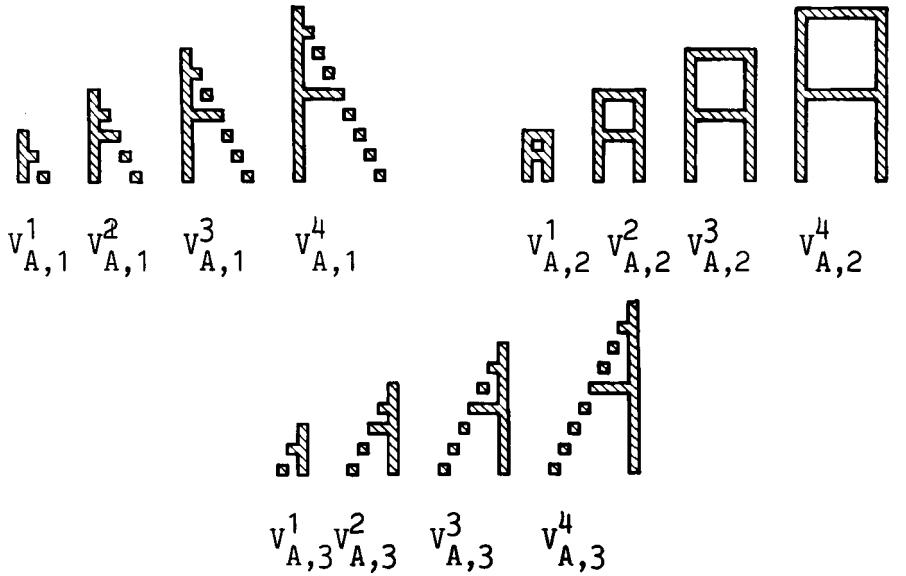


Fig. 5 Different well-written versions $V^j_{A,k}$ (k = 1,2,3; j = 1,2,3,4) for the character A

If we can find a value of k within the range $(1, k_\alpha)$ for which the expression $\hat{E}(p, i, j, \alpha, k)$ is true, we can say that 'cell p sees at least one of the sample versions of the character α in m^i and can write the corresponding E expression as $\hat{E}(p, i, j, \alpha)$. Then, corresponding with the previous notation, we write $\hat{e}(i, j, \alpha)$ for the set of cells p for which this is true and $\hat{n}(i, j, \alpha)$ for the number of cells in this set.

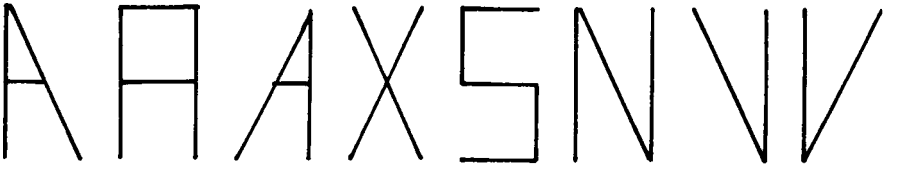


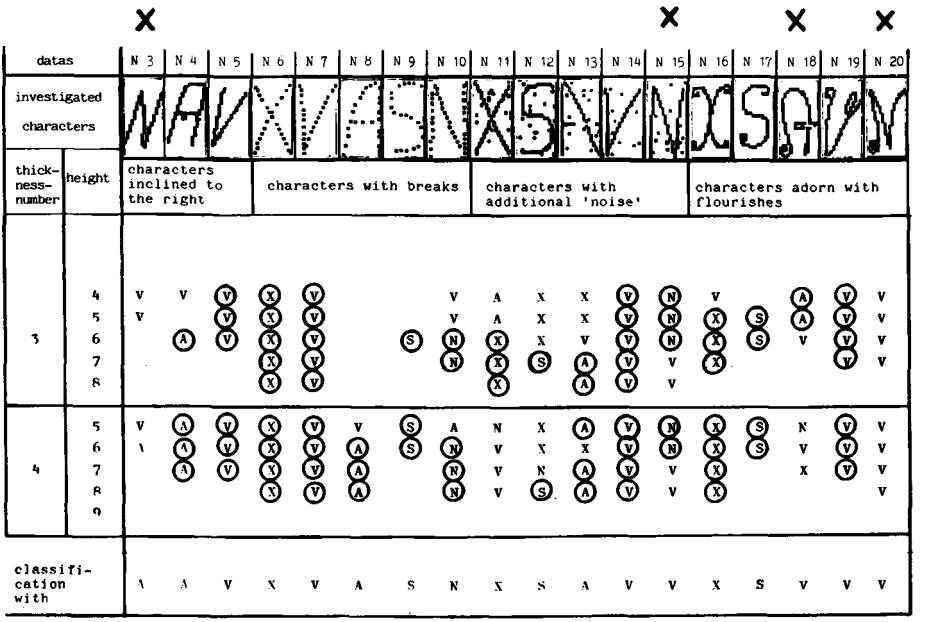
Fig. 6 The example alphabet A, X, S, N, V with three well written versions for A, 2 for V and 1 for X, S and N

However, a correction must be applied to the numbers $\hat{n}(i, j, \alpha)$ before they can be used for identification. This is because a polygonal curve $V^j_{\alpha,k}$ may contain as a sub-quantity a polygonal curve $V^j_{\beta,k}$: for example, the letter V is contained within the letter N. In this case the number of hits recorded with the character β must be corrected downwards to specify the number of 'real' β and not just the β parts of an α. In the absence of a better theoretical basis we simply subtract the number

of hits with α, giving the corrected value:

$$\hat{n}'(i, j, \beta) = \hat{n}(i, j, \beta) - \hat{n}(i, j, \alpha)$$

Where this possibility does not exist we take $\hat{n}'(i, j, \beta) = \hat{n}(i, j, \beta)$



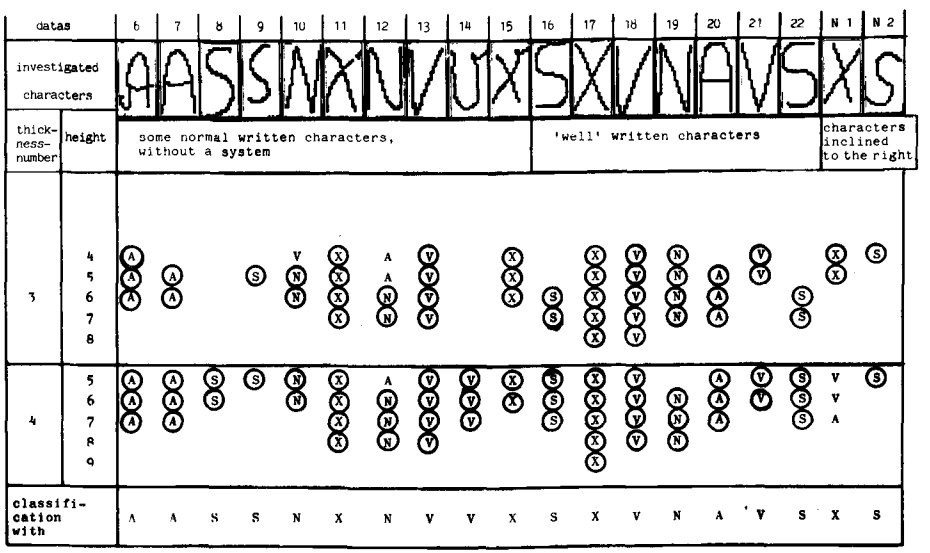


Table 3 Classification according to rule R': Correct individual decisions are marked with a circle (O), false total classifications with a cross (X)

Next we scale the number of hits for a character by dividing by the number of alternative samples, giving a final corrected value:

$$\hat{n}''(i, j, \beta) = \hat{n}'(i, j, \beta)/k\beta$$

and use the numbers $\hat{n}''$ in place of n in Rule R, which we now call Rule R′.

We have tried out this modified rule for the alphabet A,X,S,N,V . Fig. 6 shows that for A there were three sample versions, two for V and one for each of X,S and N N. Table 3 shows the individual decisions for the cases $i = 3$, $j \geqslant 4$ and $i = 4$, $j \geqslant 5$ for a series of input patterns, as well as the total decisions where Rule R′ was applied. Here as well the result is surprisingly good. Note that even the sloping A and V in N4 and N5 were correctly classified, whereas the sloping N in N3 was wrongly classified as V. This could be due to the fact that although there were sloping versions of A and V there was no such version for N.

If cases 19 and N15 of Table 3 are compared, it seems that the failure of the rule in the case of N15 was due to the fact that as well as the slight slope there was a fair amount of noise. The algorithm obviously is not capable of managing both at once. Here the deviation from the ideal form was probably too much.

It is clear that the results from Table 3 are not by themselves sufficient to prove the applicability of the rule R′ to a larger alphabet. They do show however that the idea of allowing several sample versions of each single character, and of taking numbers such as $\hat{n}''(i, j, \alpha)$ as the basis of decision, points the way to a promising solution. Rule R′ must perhaps be modified further. Tests using the whole alphabet are vital, but this must be left until later. We should like to look now at the way in which our theory has been put into practice. It will become clear that the process needs considerable computing power.

3 Details of the procedure: Serial – Parallel

For comparison purposes we assume that the hand-written characters m are in the form of a 16 x 32 binary matrix, stored in the bottom left-hand corner of a 64 x 64 binary matrix which is otherwise set to zero.

The i–times thickened version of the input pattern m is stored in the bottom left-hand corner, with dimensions (16 + i) x (32 + 2i) as shown in Fig. 7; thus m^3 and m^4 are completely contained in a bottom left-hand rectangle of dimensions 20 x 40. For these values of i, every cell which is able to see a version $V^j_{\alpha, k}$ always lies within the 64 x 64 matrix; while every sample version is contained inside a $(2j + 1) \times (4j + 1)$ rectangle and, in our sample alphabet, comprises at least $6j + 1$ points.

No program that computes the numbers $n''(i, j, \alpha)$ can manage without computing the predicates $E(p, i, j, \alpha, k)$. For a cell $p = (l, m)$ this is true if, and only if, $1 \leqslant l \leqslant 16 + i - 2j - 1$ and $1 \leqslant m \leqslant 32 + 2i - 4j - 1$. A test on E for this region requires at least $6j+1$ separate tests. Thus to calculate E for all the cells affected, at

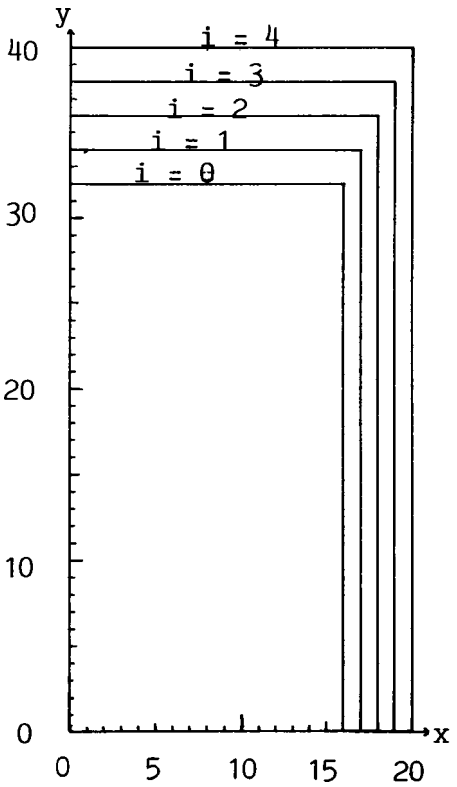


Fig. 7 Upper bounds for the admitted input patterns m and their thickend modifications m^i ($i = 1, 2, 3, 4$)

least $(16 + i - 2j - 1) \times (32 + 2i - 4j - 1) \times (6j + 1)$, or $z(i, j)$ in short separate evaluations are necessary. This has to be done for $i = 3$, $4 \leqslant j \leqslant 8$ and $i = 4$, $5 \leqslant j \leqslant 9$, giving a total of about 25000 evaluation for each sample version. Table 4 gives the detailed numbers.

In our case there are eight sample versions, so about 200,000 evaluations of the predicate are needed. So far no allowance has been made for administrative overheads and this would certainly mean a factor of from three to 30, depending on how the binary matrix was stored. Thus to calculate the numbers $n''(i, j, \alpha)$ on a serial computer for our sample alphabet of five letters needs at least one million individual operations.

The problem looks quite different if it is programmed for a bit-plane computer such as the ICL DAP. It is not strictly necessary to know about the hardware or Assembler structure of DAP to understand what follows; the reader who is interested can consult References 4 and 5. DAP is programmed using a FORTRAN extension, DAP-FORTRAN[6]. For our purposes, all we need to know is that this

i	j	z(i,j)
3	4	4725
3	5	3689
3	6	2405
3	7	1161
3	8	245
4	5	5301
4	6	3885
4	7	2365
4	8	1029
4	9	165
		24960

Table 4 The interesting combinations (i,j) and their corresponding numbers z(i,j)

offers additonal facilities to FORTRAN, whereby LOGICAL can be used to define Boolean variables and INTEGER (,), REAL (,), . . . LOGICAL (,) to create integer, real, . . . Boolean matrices of size 64 x 64 with indices going from 1 to 64. We describe a Boolean matrix with these dimensions as a *bit plane*. The following operations, amongst others, are used with such bit-planes and are of particular interest to us; here X, Y, Z are bit-planes and TRUE, FALSE are identified by 1, 0 respectively:

Local Boolean operations .AND. .OR. .XOR.

Ex: X = Y.OR.Z replaces the normal FORTRAN DO-loops

```
      DO 8    I = 1,64
      DO 8    J = 1,64
      X(I,J)  = X(I,J).OR.Z(I,J)
8     CONTINUE
```

Shifting operations SH Cyclic (C) shifts; Planar (P) shifts in the North (N), South (S), East (E) or West (W) by stated numbers of steps.

Ex: X = SHNP (Y,9) has the effect of shifting the matrix Y 9 steps to North and filling in the spaces with zeros.

This is equivalent to the FORTRAN segment

```
      DO 5    I = 1,64
      DO 5    J = 1,9
      X(I,J) = 0
5     CONTINUE
      DO 6    I = 1,64
      DO 6    J = 10,64
      X(I,J) = Y(I,J-9)
6     CONTINUE
```

The instruction ANY(X) provides a Boolean value which is TRUE if and only if X contains at least a single 1.

The instruction SUM(X) provides the number of 1's in X, as an integer.

The local Boolean operations generally result in 1 or 2 Assembler instructions; shifts of length k are done in k instructions, ANY in 2 and SUM, which is available as a subroutine, in about 250. One Assembler instruction takes about 0·4μs.

The Appendix gives a DAP-FORTRAN program for calculating the numbers $\hat{n}(i, j, \alpha)$ for $i = 3, j = 4,5,6,7,8$ and $i = 4, j = 5,6,7,8,9$ for our model alphabet with the sample versions of Fig. 5. For clarity we have not given the computation of the $\hat{n}''(i, j, \alpha)$. At the beginning of the program the input pattern m is placed in the bottom left-hand corner of the bit-plane (logical variable) LETTER as shown in Fig. 7. At the end of the program the values of $n(i, j, \alpha)$ are found in the integer variables COUNT α $(i+1, j)$. The execution time for the program on DAP is approximately 12 ms (corresponding to about 30,000 instructions) of which only about 2 ms (5,000 instructions) is for computing the $\hat{E}(p, i, j, \alpha)$ and the rest, the great majority, is for computing the sums using the operation SUM. This is the total time, including all overheads; and as this was an exploratory study we did not attempt to minimise the time further. How then was such a small number of steps in the calculation of the predicates $\hat{E}$ achieved?

The program consists of two nested DO loops. In the outer loop, the I-loop, the input character m is thickened successively by the template T to produce m^0 $(= m)$, m^1. m^2, m^3, m^4. At the beginning of each loop (CHECKPOINT 1) the variable LETTER contains m^I and at the end, (CHECKPOINT 6), by various SHIFT and OR operations, contains m^{I+1} where $I = I - 1$. The bit-plane organisation of DAP makes it possible for the thickening process to be performed for all cells simultaneously.

In the inner loop, the J-loop, the logical functions (predicates) $\hat{E}(p, i, j, \alpha)$ are computed in parallel for all cells p. The computation has been considerably simplified by observing that the various sample characters have some lines in common and that these lines grow linearly with j.

Fig. 8 shows the 10 fundamental lines l_r (r = 1, 2 . . 10) on a 3 x 5 lattice. Each l_r is an ordered set of cells defined relative to the sample origin (0, 0) and having initial and final off-sets a_r, b_r respectively. l_r consists only of those cells on the straight line joining the start point a_r and the end point b_r. Thus l_r can be taken as a set operator which when applied to a lattice cell p gives a set of cells, denoted by $l_r(p)$, which lie on the line joining $p+a_r$ and $p+b_r$.

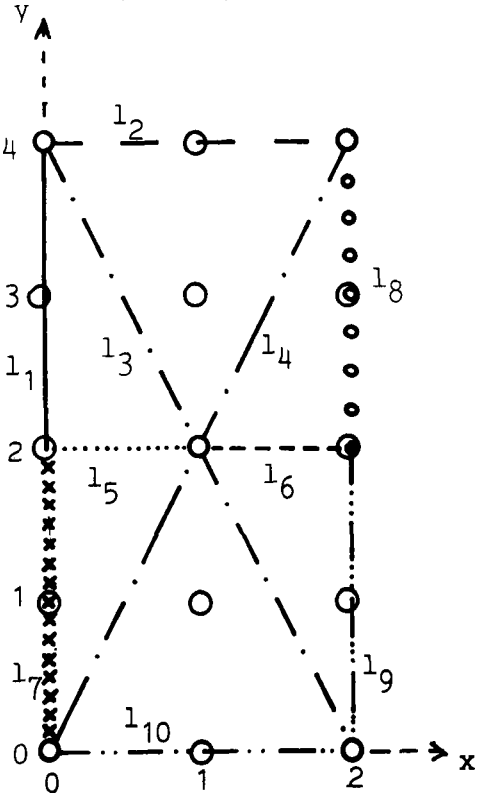


Fig. 8 The (discretised) lines $l_1, l_2, \ldots, l_{10}$

The program uses 10 bit-planes (logical variables) SM1, SM2 . . . SM10 to test for the presence of the 10 fundamental lines. At any step the variable LETTER contains m^i. If LETTER is shifted by each inverse of the off-sets in l_r in turn and then a logical .AND. is performed between SMr and the shifted values of LETTER, a cell p of SMr will be unity if an only if (iff) the line l_r is totally contained within m^i. Formally:

$$\mathrm{SMr}(p) = 1 \text{ iff } l_r(p) \subseteq m^i$$

We now define the set $(j \otimes l_r)(p)$ as the set of cells (points) on the line joining $p + j.a_r$ and $p + j.b_r$. Clearly $(1 \otimes l_r)(p) = l_r(p)$ and we have the recursive relationship, for $j \geq 1$,

$$((j + 1) \otimes l_r)(p) = (j \otimes l_r)(p + a_r) \cup (j \otimes l_r)(p + b_r)$$

This powerful relationship means that if we have the results computed at the end points of the line $j \otimes l_r$ we can combine these to get those for $(j + 1) \otimes l_r$.

This permits us to compute the results SMr recursively for the various values of j. If the values of SMr(p) are known for some value of j, then if the operation

* "shift SMr by the vectors $-a_r$, $-b_r$ in turn and perform the logical .AND. of each and store the result in SM_r"

is performed, SMr(p) will be TRUE if and only if $((j + 1) \otimes l_r)(p)$ is totally contained within m^i.

The initialisation of SMr according to the above is performed at the start of the I - loop (CHECKPOINT 1) and the operation* is performed at the end of the J - loop (CHECKPOINT 5) in preparation for the next value of j.

Each letter in the sample is defined by the union of a number of lines l_r; thus for the character α of size 1:

$$V_\alpha^1 = l_{\alpha 1} \cup l_{\alpha 2} \cup l_{\alpha 3} \quad \ldots \cup l_{\alpha t}$$

where $\alpha 1, \alpha 2, \ldots \alpha t$ are selected from 1, 2, ... 10 in order to define the sample character α. The definition of $j \otimes 1$ is such that:

$$V_\alpha^j = (j \otimes l_{\alpha 1}) \cup (j \otimes l_{\alpha 2}) \quad \ldots \cup (j \otimes l_{\alpha t})$$

The statements at the start of J-loop (CHECKPOINT 2) implement these selections for three samples of the letter A (A1, A2, A3), two for V (V1, V2) and one each for N, S, X (N1, S1, X1).

The statements between CHECKPOINT 3 and CHECKPOINT 4 complete the identification of those cells p which see the character α in m^i. That is, A is TRUE only where LETTER contains any sample 'A' of size j and hence the variable A contains $\hat{E}(p, i, j, A)$. The SUM operation is then performed (CHECKPOINT 4) to fill the variable COUNT α with $n(i, j, \alpha)$. The SUM operation is relatively long, so checkes are incorporated to eliminate a call to this when this is not necessary.

We hope this short sketch of the program logic has shown the reader how the program actually computes the numbers $\hat{n}(i, j, \alpha)$. The number of instructions needed to do this can be found from the various figures given in the paper.

The calculation of the numbers $n(i, j, \alpha)$ for the (i, j) combinations which interest

us can be done in a bit-plane computer such as DAP with relatively little computing overhead, whereas in a serial computer this would take too much computer time to be practical: the ratio is probably at least 100 : 1. The total number of bit-planes necessary is

$$L + \sum_{\alpha} k_{\alpha} + n$$

where L is the number of basic lines (cf. Fig. 8) which go to make up the polygonal curve $V^{j}_{\alpha, k}$, k_{α} is the number of sample versions permitted for the character α and n is the number of characters in the alphabet. Here the number is about 20. In addition $12n$ bit-planes could be used to store the numbers $\hat{n}''(i, j, \alpha)$ but this wastes space and is not strictly necessary.

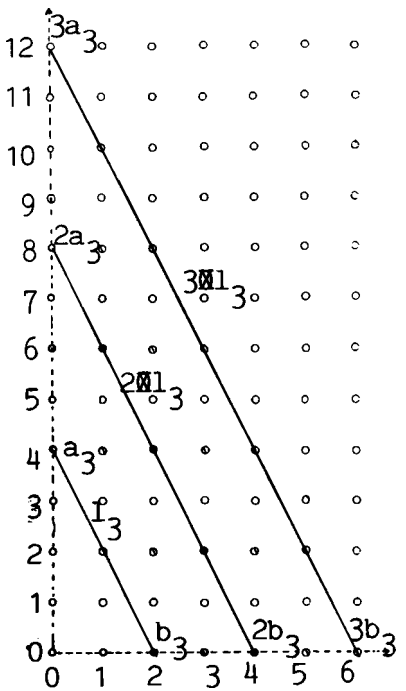


Fig. 9 The (discretised) lines l_3, $2 \boxtimes l_3$ and $3 \boxtimes l_3$

We are of the opinion that this procedure can still be speeded up by a factor of 10, if one minimises the Boolean expressions which occur and instead of the numbers $\hat{n}(i, j, \alpha)$ uses the bit-planes $\hat{e}(i, j, \alpha)$ and $\hat{e}(i, j, \alpha, k)$ directly to bring about a decision. Also a simple thickening procedure could be developed which lets the useful part of the hand-written character grow, but not the noise.

4 Conclusion

The method we have described, in which the automatic recognition of hand-written characters is done by first 'thickening' the input pattern to a fair extent (by about

30% of the height of the input pattern) and then looking for the number of well-written versions it contains (approximately of the size of the input pattern), has led to results which, in view of the simplicity of the procedure, are very positive and encourage us to further research in this direction. The method was developed for bit-plane computers and runs very fast and efficiently on the DAP. A direct conversion to serial computation would lead to impossibly long run times. Our method still needs a proper theoretical foundation, however: the present basis is purely intuitive.

Acknowledgments

The authors are very grateful to Professor R. Vollmar of Brunswick University for the various and fruitful discussions about this work. They would also like to thank Professor D. Parkinson and Vic Green of Queen Mary College for making it possible for the second author to carry out the practical work necessary on the DAP at Q.M.C. Our thanks are due to H.A. Morris of ICL for translating the paper from German and for a number of useful suggestions; also the (unknown) referee and to the editor for improving the text.

References

1 ROSENFELD, A. and KAK, A.C.: *Digital picture processing,* New York 1976.
2 NIEMANN, H.: *Methoden der Musterkennung,* Frankfurt 1974.
3 PECHT, J.: 'Ein neuer Ansatz zur Erkennung handgeschriebener Zeichen mit Hilfe DAP-ähnlicher Feldrechner', Technical University, Brunswick, May 1980.
4 PARKINSON, D.: 'An introduction to array processors', *Systems International,* November 1979.
5 ICL: VME/B/DAP: APAL Language, Technical Publication 6919, London 1979.
6 ICL: DAP: Introduction to FORTRAN-Programming, Technical Publication 6755, London 1978.

Appendix: Program for character recognition

```
      Entry subroutine CHARACTER_RECOGNITION
C
C     global declarations for the input plane (LETTER) and the integer matrices
C     for output dates (COUNTalpha (alpha = A, N, S, V, X)):
      common/data/LETTER
      common/data2/COUNTA, COUNTN, COUNTS, COUNTV, COUNTX
      integer COUNTA(,), COUNTN(,), COUNTS(,), COUNTV(,), COUNTX(,)
C
C     planes for storage of the recognition bit patterns of the lines l1-l10:
      logical SM1(,), SM2(,), SM3(,), SM4(,), SM5(,), SM6(,), SM7(,),
     1 SM8(,), SM9(,), SM10(,)
C
C     plane for intermediate storage to shorten the line computing expressions
      logical SHILF(,)
C
C     planes for the storage of the recognition bit patterns for the single versions
```

```
C       of the characters:
        logical A1(,), A2(,), A3(,), N1(,), S1(,), V1(,), V2(,), X1(,)
C
C       planes for the recognition bit patterns of the characters:
        logical A(,), N(,), S(,), V(,), X(,)
C
C       initialisation of the output integer matrices:
        COUNTA = 0
        COUNTN = 0
        COUNTS = 0
        COUNTV = 0
        COUNTX = 0
C
C       begin of the processing:
C
C       begin of the I-loop (I-1 is the number of times the input pattern LETTER is
C       thickened):
        do 100 I = 1,5
C
C       initialisation of the line recognition bit patterns (CHECK-POINT 1):
        SM7 = shsp (LETTER, 1) . and . shsp (LETTER 2) . and . LETTER
        SM1 = shsp (SM7, 2)
        SM8 = shwp (SM1, 2)
        SM9 = shwp (SM7, 2)
        SM10 = shwp (LETTER, 1) . and . shwp (LETTER, 2) . and . LETTER
        SM2 = shsp (SM10, 4)
        SHILF = shsp (LETTER, 2)
        SM5 = SHILF . and . shwp (SHILF, 1)
        SM6 = shwp (SM5, 1)
        SM3 = shsp (LETTER, 4) . and . shsp (shwp (LETTER, 1), 2) . and .
     1  shwp (LETTER, 2)
        SM4 = LETTER . and . shsp (shwp (LETTER, 1), 2) . and .
     1  shsp (shwp (LETTER, 2), 4)
C
C       begin of the J-loop (J is the heigth of the model letters looked for):
        do 150 J = 1,9
C
C       setting up the recognition bit patterns for each model letter version of height
C       J (CHECKPOINT 2):
        A1  = SM1 . and . SM3 . and . SM5 . and . SM7
        A2  = SM1 . and . SM2 . and . SM5 . and . SM6 . and . SM7 . and . SM8 . and
              . SM9
        A3  = SM4 . and . SM6 . and . SM8 . and . SM9
        N1  = SM1 . and . SM3 . and . SM7 . and . SM8 . and . SM9
        S1  = SM1 . and . SM2 . and . SM5 . and . SM6 . and . SM9 . and . SM10
        V1  = SM3 . and . SM8 . and . SM9
        V2  = SM1 . and . SM4 . and . SM7
        X1  = SM3 . and . SM4
C
```

```
C       cumulating the recognition bit patterns of the different versions for each
C       model letter (CHECKPOINT 3):
        A   = A1 . or . A2 . or . A3
        V   = V1 . or . V2
        S   = S1
        N   = N1
        X   = X1
C
C       summing up the bits of the recognition bit patterns of each model letter if
C       the corresponding combination (I,J) is of interest (CHECKPOINT 4):
        if ((I.lt.3) . or . (J.lt.4)) goto 120
        if (any (A)) COUNTA (I,J) = sum (A)
        if (any (N)) COUNTN (I,J) = sum (N)
        if (any (S)) COUNTS (I,J) = sum (S)
        if (any (V)) COUNTV (I,J) = sum (V)
        if (any (X)) COUNTX (I,J) = sum (X)
C
120     continue
C       enlargement of the lines looked for (CHECKPOINT 5)
        SM1 = shsp (SM1,2) . and . shsp (SM1, 4)
        SHILF = shsp (SM2, 4)
        SM2 = SHILF . and . shwp (SHILF, 2)
        SM3 = shsp (SM3, 4). and .shwp (SM3, 2)
        SM4 = SM4 . and . shsp (shwp (SM4, 2), 4)
        SHILF = shsp (SM5, 2)
        SM5 = SHILF . and . shwp (SHILF, 1)
        SHLIF = shsp (SM6, 2)
        SM6 = shwp (SHILF, 1) . and . shwp (SHILF, 2)
        SM7 = SM7 . and . shsp (SM7, 2)
        SHILF = shwp (SM8, 2)
        SM8 = shsp (SHILF, 2) . and . shsp (SHILF, 4)
        SHILF = shwp (SM9, 2)
        SM9 = SHILF . and . shsp (SHILF, 2)
        SM10 = SM10 . and . shwp (SM10, 2)
C
150     continue
C       end of the J-loop
C
C       spreading the input letter once more (CHECKPOINT 6):
        SHILF = LETTER . or . shnp (LETTER, 1) . or . shnp (LETTER, 2)
        LETTER = SHILF . or . shep (SHILF, 1)
C
100     continue
C       end of the I-loop
C
        return
        end
```

Hardware design faults: A classification and some measurements

T.L. Faulkner, C.W. Bartlett and M. Small
ICL Mainframe Systems Development Division, Manchester

Abstract

This paper analyses the types and quantities of logic hardware faults found and corrected during development commissioning, system validation, production introduction and field support for an MSI implementation of the mainframe of a medium range computer system. It is found that the faults can be classified into fault types which, with particular exceptions, apply generally to all units of the mainframe design and that there is a variation of the ranking of the fault types as the development progresses. The results show that most design faults are found while the prototype subsystems are being commissioned in isolation and that those faults which are not found until the complete system is brought together are more complex, concerning tolerancing, specifications and concurrency. The implications of this information with respect to the design of VLSI systems is discussed and some improvements to design methodology are proposed.

1 Introduction

The motivation for the work described in this paper derives primarily from the realisation that the imminence of a move from an MSI to a VLSI product basis would require a change in the traditional strategy for the development of hardware products.

In the past, hardware has often been developed on the basis of designing to a reasonable standard, the early creation of a prototype, and the testing out of design faults before full scale production was committed (though not before production introduction was started). This approach had been indicated by the relatively high ratio of initial build time to modify time and a management desire for a firm adherence to such milestones as 'switch on prototype'.

With the incident of VLSI, the ratio of initial build time to modify time begins to approach unity and hence the risks to timely achievement of the programme are much more dominated by the incidence and nature of design faults. The development strategy required for VLSI products is therefore one of prevention and early discovery of design faults with an expectation of minimal interference with timescales once the silicon has been committed.

With this in mind, a detailed recording and analysis of the design faults actually encountered during the life of a typical product was commissioned with the

objective of determining those areas where improvement was necessary prior to the move to VLSI technology.

The product for which these data were collected is a mid-range mainframe with principal units: Store, Information Interchange Control, I-O Control, Order Code Processor, and Maintenance Processor, (see Fig. 1).

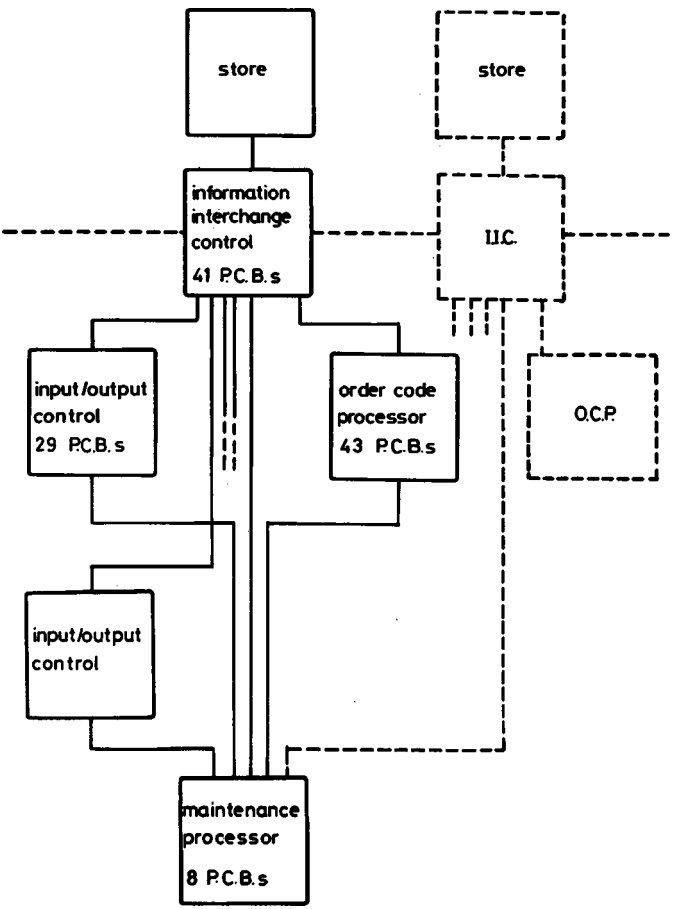


Fig. 1 Structure of product

The technology used was Schottky TTL MSI mounted on multilayer printed circuit boards (p.c.b.s) each carrying approximately 120 integrated circuits.

The development environment was one in which there was extensive Design Automation support covering such areas as physical layout, fan out/in control, timing, power loading and thermal loading. There was no logic simulation and no mechanised control of interface specification. The main means of controlling the design integrity was a system of independent desk checking coupled with formal peer group reviews and external audits.

The design strategy was to attempt to map closely functional and physical partitions with approximately a 25% investment in error management hardware.

Organisationally, the development consisted of a central design group controlling the functionality of and interfaces between the principal units with separate projects being responsible for the detailed design of these units. The development strategy was to build a prototype of each unit and to commission each of these in isolation, as far as was practical, then to construct and commission a complete system from prototype units.

The development work for which data were collected covered a four year period. This period started with the initial development of the basic subsystems and proceeded in eight major stages of approximately equal duration, through the introduction of various enhancements such as the attachment of new peripheral couplers, performance options and increased configuration sizes until all major subsystems and facilities were validated and into production.

In this paper we present the structure adopted for the collection of the data and the results obtained. These results are discussed and the implications for the move to VLSI are given. The steps taken to implement the implied requirements are described and finally, an indication of the success of these moves is given in the context of a pilot translation of an existing design.

2 Fault Classification

The fault classification used here was chosen to enable data concerning design faults actually occuring in the existing environment to be collected cheaply and easily. The actual classification, while being reasonably abstract, is biassed towards the type of product and development environment described above. This bias is felt to be valuable since it matches practical engineering experience and makes the concepts more accessible.

The classification is hierarchical in concept, and classes may be subdivided at any level. In practice the maximum depth used was 4. The major fault classes and their subdivisions are:

- Specification faults
- Environment faults
  - Technology faults
  - Design rules broken
  - Noise
  - Design Automation faults
- Realisation Faults
  - Logic faults
    - Simple
    - Exception conditions
    - Concurrency

Time outs
Initialisation
Clock faults
Performance and size faults
Error management faults
Testability/Maintainability faults
Compatability

2.1 *Specification faults*

This is an important group of design faults covering errors, deficiencies and ambiguities in the design specifications. These faults can be very expensive to correct if they are found too late. The most expensive, and hence interesting, specification faults are those where it is not realised until a late stage of the development process that the system has been incorrectly partitioned.

2.2 *Environment faults*

These faults result from the physical constraints or environment in which the actual design is implemented. For example the logic family used will have certain characteristics, some of which will not be wanted, and will impose various constraints. If these are not correctly taken into account a logically correct design may not work at all or will only work unreliably.

2.2.1 Design rules broken: It is normal to deal with the constraints associated with the logic family and interconnection methodology by imposing a set of 'design rules'. Failure to follow these rules may result in worst-case timing not being catered for, glitches or race conditions, overheating, overloading and unreliability. We included under this category a small number of instances where the design rules were changed to accommodate a reduction in the performance of the actual components.

2.2.2 Noise: This can arise particularly on long wires and tracks and so it tends to affect those areas of the design where long tracks are more likely, for example clock circuits, reset lines, long data highways, parity trees and handkey/lamp circuits. It is usually due to non-observance of design rules. Detection can be by continuous or impulse electromagnetic compatability (EMC) tests.

2.2.3 Design Automation faults: These include faults directly introduced by the design automation system as well as those which should have been detected by the system but which were not. Gross errors are easily spotted but obscure ones can slip through manual checks.

2.3 *Realisation faults*

These are the various faults which arise in the implementation of a specification. The faults are classified according to their nature where they concern the major required function of the subsystem, or according to the part of the subsystem containing the fault where they do not. These classifications are logical errors,

performance problems, testability and maintainability problems, clock faults, error management faults.

2.3.1 Logic faults: These were classified according to type.

Simple logic fault. This is a straightforward fault in the logic so that it does not meet the specification.

Concurrency faults. These are logic faults which only become manifest when certain asynchronous signal patterns occur. They are seen as reduced reliability under certain work patterns. They can be very difficult and expensive to find.

Time-out faults. These concern system module interactions: experience shows that the designed times for time-outs and repeat-action delays are sometimes incorrect. This can lead to performance degradation or system crashes.

Initialisation. These concern the correct initialisation of the machine state at switch-on or after reset. Since the machine states may be correct by chance or may not always be relevant these faults may appear to be intermittent and can be difficult to find.

2.3.2 Clock faults: These concern the design of the clock circuits. They include a mixture of electronic and logical faults. An important subdivision of these faults was found to be resynchronisation after handling asynchronous signals by clocked logic. Other faults may concern correctly starting and stopping the clock, 'hiccup' conditions, design of tapping circuits, tolerancing, phase-locked oscillators and scan-in scan-out testing requirements.

2.3.3 Performance and size problems: These are faults where the functional requirement is met except for performance or size. They can apply to most subsystems including microcode and firmware. Phased development is particularly prone to discover size problems at late stages.

2.3.4 Error management faults: These concern the design of the detection, containment and recovery from error conditions.

2.3.5 Testability and maintainability problems: These are where the design meets all requirements except those concerned with testability or maintainability.

2.3.6 Compatibility faults: These concern the correct working of a subsystem in some but not all of the system environments specified.

3 Results

The results obtained are summarised in Figs. 2 to 5.

Fig. 2 shows for the 3 major subsystems the average numbers of faults found per p.c.b. and the phase of development in which they were found. It can be seen that there were nearly twice as many faults per p.c.b. found for the O.C.P. than for

either of the other two subsystems. However it is difficult to assess the significance of this difference since complexity is not explicitly taken into account by the measure, faults per p.c.b. A better metric would relate faults found to complexity in a similar manner to the models of faults in software.[1] The proportion of faults found during independent commissioning of the prototype subsystems is also different for each subsystem, the smallest proportion being found before system integration for the Store and Information Interchange Control subsystem. This subsystem has a complex asynchronous interface with the other two, making it extremely difficult to test comprehensively in isolation.

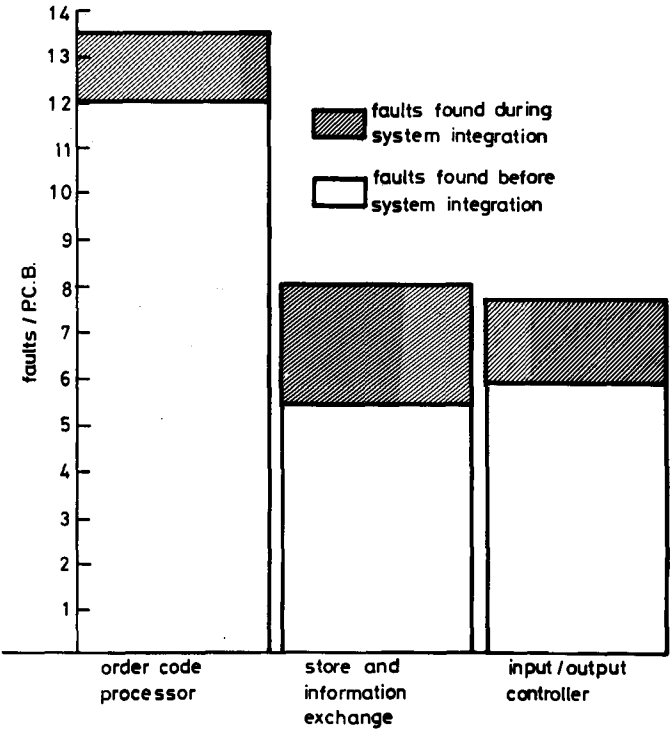


Fig. 2 When faults are found

Figs. 3 and 4 show the proportions of the different classes of faults found before and during system integration. These show a variation in the profile of faults at the different stages. This suggests a variation in the detectability of the different classes. While the subsystems were being commissioned in isolation over half the faults found were classified as logic faults, and three quarters of these were considered to be simple logic faults. Specification faults and technology rule violations accounted for only 10% of faults found. During system integration technology rule violations were the largest class of faults found, with specification faults and technology rule violations accounting for 44% of the total. Logic faults were a smaller proportion, 21%, and these were predominantly complex (concurrency, exception conditions etc.).

The discovery of technology rule violations reached a peak when volume production was achieved, i.e. when the design was exposed to the full range of component variation. There were peaks in the discovery of specification faults coincident with the first attachment of new subsystems. This shows the difficulty of correctly specifying and designing interfaces, the 'unusual' and end-effect conditions associated with I/O being a particular problem.

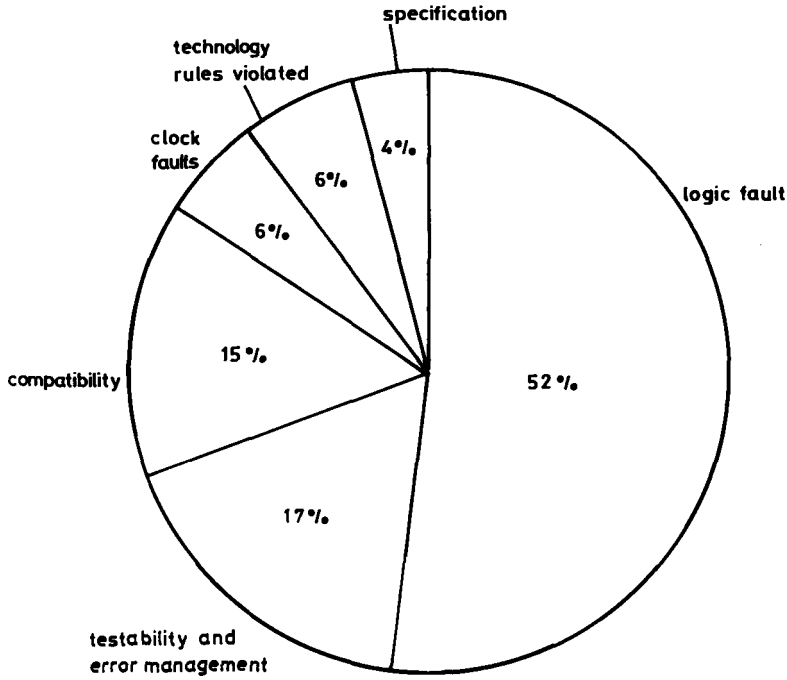


Fig. 3 Breakdown of types of faults found BEFORE system integration

In general design faults remain latent until the right combination of subsystems is tried, so that, for example, error management facilities are extremely difficult to debug and in practice required the use of the system software to expose problems.

For the system described here one design fault led on the average to one change in the modification level of one p.c.b.

4 Improved design methodology

The information obtained from this study can be used to improve the hardware development methodology. The approach taken was the careful application of known techniques directed at the problem areas uncovered.

The first important factor is that design management create an environment which supports an emphasis on design quality. This is important to direct the considerable skills of the development staff onto this question. Many of the faults observed

stemmed from a misconception of the required trade-offs between performance, cost, timescale and quality by staff working at the detail level.

Examination of the instances of specification faults showed them to be caused by a generally relaxed attitude towards the control of specifications at all levels and a general inability to readily perceive the inter-relationships between the various specifications.

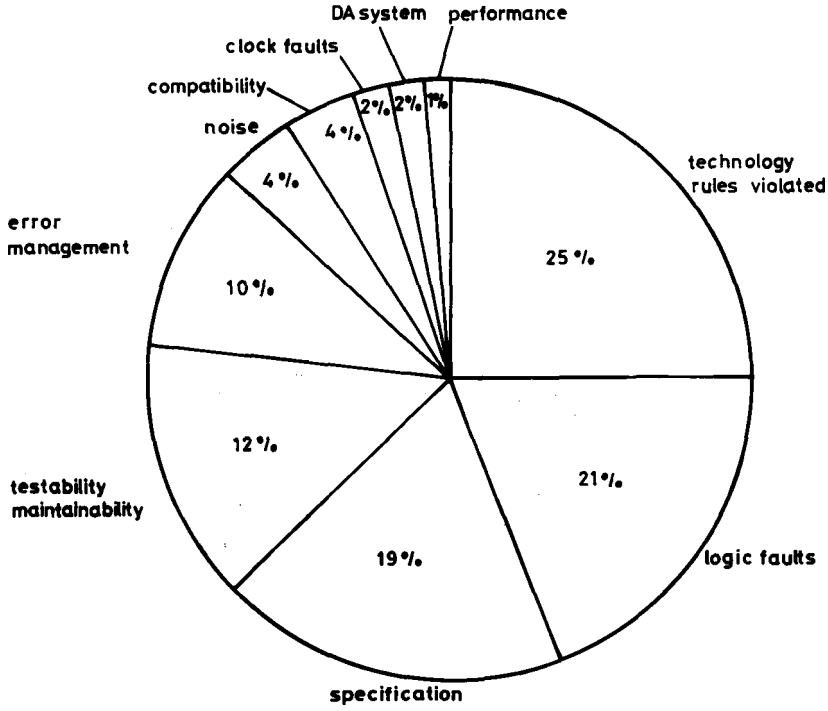


Fig. 4 Breakdown of types of faults found DURING system integration

The solution to these problems can be found in the Software Engineering discipline used within ICL. This is a structural modelling system[2] which imposes strict formality on the specification of interfaces and functions and allows the inter-relationships between modules to be readily tracked. This software system was modified to provide an analagous service with respect to hardware design.

For the faults which arose in the realisation of the design two remedies were considered. These were 'proof of correctness' and simulation.

The 'proof of correctness' method is not proposed since it was felt to pose too high a risk, bearing in mind the current state of the art, and because it was thought that it represented too great a change in the methods used by the hardware engineers.

The method actually adopted was that of simulation and a comprehensive low-level interactive simulation system[3] was commissioned and is now in use. This enables

who actually did the work described in this paper, and in particular to D.W. Ashcroft.

References

1 KITCHENHAM, B.A.: 'Measures of program complexity', *ICL Tech. J.*, 1981, **2** (3), 298-316

2 McGUFFIN, R.W., ELLISTON, A.E., TRANTER, B.R. and WESTMACOTT, P.N.: 'CADES - software engineering in practice', *ICL Tech. J.*, 1980, **2** (1), 13-28.

3 WHITTAKER, R., BROOKE, A., PIERIS, D. PIERCE, D.H.: 'Simulation at universal level - a user guide', ICL Internal Document AMDS/SAUL.

4 WILLIAMS, M.J.Y.: 'SIMBOL 2 - a logic description and simulation language', ICL Internal Document DA 467.

5 WILLIAMS, M.J.Y. and McGUFFIN, R.W.: 'A high-level design system', *ICL Tech. J.*, 1981, **2** (3), 287-297.

6 FAGAN, M.E.: 'Design and code inspections to reduce errors in program development', *IBM System J.*, 1976, 15 (3).