ICL TECHNICAL JOURNAL

Volume 7 Issue 1 May 1990

Published by
International Computers Limited
at
Oxford University Press

ICL
AN STC COMPANY

ICL TECHNICAL JOURNAL

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

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

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

1990 subscription rates: annual subscription £40 UK, £44 EEC, £48 rest of world, US $95 N. America; single issues £22 UK, £25 EEC, £27 rest of world, US $52 N. America. Orders with remittances should be sent to the Journals Subscriptions Department, Oxford University Press, Pinkhill House, Southfield Road, Eynsham, Oxford OX8 1JJ.

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

ICL TECHNICAL JOURNAL
Volume 7 Issue 1

Contents

Obituary

PROFESSOR COLIN GOODMAN

With great sadness we have to say that Professor C.H.L. Goodman, a member of the Editorial Board since the beginning of 1987, died of leukaemia on 7 January 1990 at the early age of 62. His illness had been diagnosed a little under a year ago but he had seemed to be responding well to chemotherapy and in fact, although in hospital for most of the time, he was working with his characteristic energy and incisiveness up to the middle of December. Knowing this, we were even more shocked by the news of his death.

Colin Goodman, F.Inst.Physics, F.Royal Inst. of Chemistry, had a distinguished career in materials science. He read chemistry at Oxford, where he did postgraduate research in the Metallurgy Department. He spent 10 years in the GEC Research laboratories at Wembley, where he made pioneering investigations into semiconductors, ultra-pure materials and high frequency silicon transistors. He joined Standard Telephone Laboratories at Harlow (now formally named STC Technology Limited) in 1960, where he set up a range of research projects concerned with electronics applications of high technology materials, and incidentally initiated and ran a flourishing seminar series. He held visiting professorships at Chelsea College of Science and Technology from 1970, at the University of Warwick from 1977 and at King's College, University of London from 1987. He had a deep and life-long interest in the properties of materials and an encyclopedic knowledge of the field; latterly his imgination had been fired by the recently discovered phenomenon of superconductivity in certain ceramic materials at temperatures far higher than had previously been thought possible.

We shall miss him greatly, both as a very positive contributor to our discussions in the Editorial Board and as a very individual personality.

Editorial

This issue of the Journal is devoted to two subjects: the recently announced DRS6000 series and interfaces of various kinds both to and within computers. The three papers on DRS6000 are introduced by Ed Parton. A few general remarks follow on interfaces, a subject that has interested the editor for many years.

Many different objectives may be met by including interfaces in computer systems. A supplier, for instance, may find them valuable because they allow a greater variety of market needs to be satisfied by fewer basic hardware or software components, and because they permit parallel but independent development of those components by separate teams, and also allow products to be enhanced individually.

Equally, interfaces appeal strongly to purchasers because they believe that, properly standardised, interfaces can: give better assurance of the interchangeability or interworking buyers may want: allow them to build systems out of components from a wider variety of suppliers: simplify managing the development of an application by speeding and cheapening it: and, finally, contribute to comprehension of the system as a whole and so facilitate staff training. Interfaces in products designed to established international standards can also permit independent verification of conformance by third parties.

Fundamentally, these purposes are all served by interfaces because they cater either for *interworking* between system components from various sources, or for *interchangeability* on the same side of an interface between components having similar or identical functionality. It is interesting that these concepts of interworking and interchangeability were firmly established in telecommunications practice long before computers appeared.

The art and practice of defining, standardising and designing interfaces between elements composed solely of hardware is reasonably well established. The kinds of difficulties that still arise with them are more often political or commercial than technical. Appropriate procedures for interface design and standardisation in telecommunications engineering have evolved over perhaps a century. Certainly there are many hardware interfaces still in use that are technically inferior or unsuitably located but this is usually for historical reasons.

The same is not so generally true of an interface involving software or a person; here there are real scientific and technical problems left to be solved. A common difficulty is to define abstract concepts with sufficient clarity and precision for designers working in separate teams on software products intended to interwork, to have the same understanding of how their respective products must perform, and then of how to test them to ensure that they always interwork correctly. Laying down and conforming to standards is often seen as a solution. The paper by Coon talks about standards for describing documents so that they may be exchanged between different systems and that by Parker discusses standards for ways to authorise access to distributed systems.

The growing practice of allowing public interrogation of centralised computer databases, while retaining secure control over what may or may not be seen through the user/system interface, creates major intellectual challenges for administrators and technical specialists alike. Two papers in this issue on security at the user interface complement one another and also those on security that appeared in the last.

To conclude I should like to pay a personal tribute to the retiring editor Dr Jack Howlett. He has brought to the job wide computing experience and many other abilities. Allied to his mathematical and management skills is a command of languages which enabled him to translate French mathematical works into English. He was a joint editor of the History of Computing in the Twentieth Century.

Starting as a mathematician with the LMS railway and after wartime research for the government, he joined the Atomic Research Establishment at Harwell in 1948. Here he became, first head of the Computer group and then head of the independent Atlas Laboratory which ran the most powerful machine of its day, and grew into the major centre of computing research which it has become.

For more than eleven years he has edited the Journal single-handed. Indeed he was responsible for persuading the Company to start it. He has shown the keenest perception of the technical subjects that were becoming important to the Company's business and dedication to getting hold of the right authors. If they were not punctual in delivering their MS he chased them – in the most kindly way – and if they did not find the right words to express their ideas, he would revise the text himself. All this he always did with the greatest good humour.

It has been tremendously encouraging that he continues not only as a member of the Editorial Board but has also volunteered actively to help with the considerable day to day detail entailed. I am amazed that he was able to do the job single handed – and with time to spare for many other activities. The Company – as well as the present editor – has every reason to be grateful to him.

J.M.M. Pinkerton

DRS6000 UNICORN

Foreword
The DRS6000 (UNICORN) Project

ICL's commitment to the UNIX Operating System as an open standard, and in particular to AT&T's UNIX 5.0 and its successors, is now well known; as a founder member of X/OPEN and a key member of UNIX International the company's international reputation and standing as a UNIX Systems Integrator have progressively increased throughout the latter half of the 1980s.

The UNICORN project was formally approved in December 1987; this was to be a major development program, intended to increase substantially ICL's presence in the UNIX marketplace. A product centre – the Advanced Servers Product Centre – was formed in Bracknell, tasked with designing a new UNIX product, UNICORN, for manufacture at ICL's Ashton (Manchester) plant and introduction in the first quarter of 1990.

The project was set up with the specific objective of providing a UNIX system that combined state-of-the-art technology with the accepted international Open Standards. The three papers that follow in this issue of the ICL Technical Journal show how these objectives were met in the three key areas of development –

- the UNIX operating system, UNIX 5.4
- the multiprocessing hardware architecture
- the electromechanical packaging, which enabled ICL to deliver a product for sale anywhere in the world.

The UNICORN product was launched in January 1990 as the ICL DRS6000 series.

P.E. Parton
Manager, Advanced Servers Product Centre

Architecture of the DRS6000 (UNICORN) Hardware

G. Poskitt
Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berks.

Abstract

UNICORN is a multi-user, multi-processor UNIX system implemented in RISC (Reduced Instruction Set Computer) technology. Cache memories are provided to enable the very high speed of the processors to be fully exploited, the multiple processors to share, efficiently, a common memory sub-system and the I/O traffic to be buffered. The paper describes the basic architecture of the system, in particular the caching mechanism and the means by which the various physical cache memories are kept consistent among themselves and with main memory.

The UNICORN product is a development by the Advanced Servers Product Centre which is part of the Servers Product Group, which itself is part of ICL's Office Systems Division.

UNICORN is a multi-user UNIX system designed to meet the requirements for office applications for the nineties. The processing element of UNICORN is based on SPARC (Scaleable Processor ARChitecture) RISC technology.

One of the key features of UNICORN is scaleability. It has been designed to be cost effective from 32 users to several hundred. Scaleability in UNICORN includes the following:

Processor Power – UNICORN is a multi-processing system and from one to four processors may be configured.

Memory Capacity – through the use of multiple memory modules and 1 Mbit and 4 Mbit DRAM technology, memory capacities from 16 to 512 MBytes are configurable.

Disc Capacity – UNICORN allows great flexibility in the number of disc drives which may be attached. Total capacity ranges from 760 MBytes to 5 GBytes in a single cabinet with capacities of up to 15 GBytes using expansion cabinets.

I/O Capability – through the use of multiple I/O cards and expansion cabinets there is extreme flexibility in the user connectivity, communications, and networking capability that may be configured.

1 The UNICORN Architecture

A systems architecture has been defined for UNICORN that includes the CPU board set, including memory, and the I/O sub-system. It uses a dual bus architecture: the industry standard VMEbus ("Versa Module Europe" – the most popular 32-bit bus in the industry) for I/O and the 64-bit ICL proprietary HSPbus (High Speed Private) for interfacing to memory.

The architecture of UNICORN embodies certain principles which are fundamental to an understanding of the architecture.

Multiprocessing – In order to allow flexibility in configuring performance to user requirements and to provide the performance which will be required through into the 1990s, an architecture which supports true symmetrical shared memory multi-processing is mandatory. Symmetrical multi-processing allows any task to be executed by any processor in the system. This gives predictable performance which is scaleable with the number of processors. UNICORN allows up to four processors to be configured.

Industry Standard I/O bus – One of the requirements for UNICORN is as an OEM platform. To allow third parties to have the option of utilising off the shelf I/O controllers, the I/O bus is an industry standard VMEbus.

Caching – The UNICORN CPUs have caches. A cache is a very high speed local memory which contains frequently accessed instructions and operands. This is necessary to allow the SPARC microprocessor to run at full speed. Also a cache is necessary to reduce the bandwidth to main memory so that multiple processors can share a common memory subsystem. Thirdly I/O transfers also need to be buffered or cached so that I/O traffic does not interfere with and slow down CPU/Memory traffic on the High Speed Private Bus.

Coherency – The use of caching on the processors and I/O is necessary for performance. This is a hardware technique for improving performance and so should be transparent to software as much as possible. Caching potentially introduces the problem of stale data when caches become out of step (incoherent or inconsistent) with main memory and each other. The architecture, is therefore designed to provide cache coherency through hardware snoop (bus watching) logic.

These fundamental design principles lead to the conceptual architecture shown in Fig. 1.

A CPU is buffered by a cache and sits on the HSPbus. Other CPUs (up to four) are buffered and interfaced in a similar way. Memory modules also sit on the HSPbus and thus are equally accessible from all processors. This creates a symmetrical shared memory multiprocessing system.

I/O controllers sit on the VMEbus and this is linked to the HSPbus via a cache; so the I/O and CPU interfaces are logically identical.

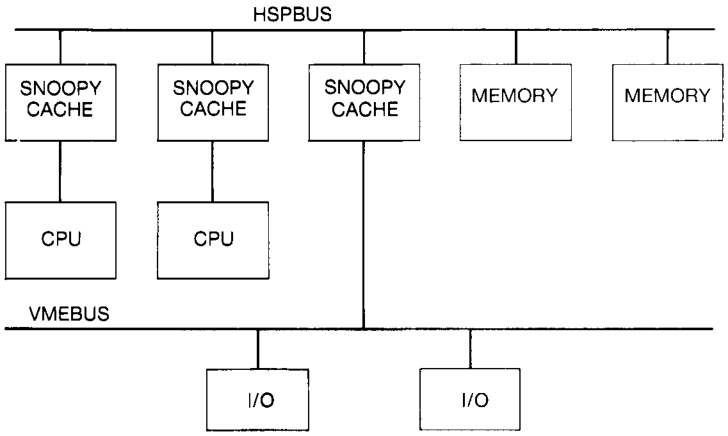


Fig. 1 UNICORN conceptual block diagram

As all modules which talk to memory are buffered by caches and interface to memory via the HSPbus, this becomes the common point for implementing cache coherency. By adding hardware snoop facilities to each and every cache, a cache can be made aware of the activities of all other caches. An algorithm can then be established to maintain cache coherency.

1.1 UNICORN implementation

This section shows how the fundamental design principles outlined in the previous section become the implementation which is UNICORN. Figure 2 shows the architecture of UNICORN where it can be seen that it is equivalent to the logical architecture already described.

The I/O caching facility appears only once in a system, so other system facilities which are required only once have been incorporated into the I/O module and it is now named the Central Services Module. An example of the facilities included are the Boot and Diagnostics functions.

The CPU modules have gained an interface to the VMEbus. The cache in the Central Services Module is only useful for VMEbus slave transfers, i.e. those transfers which are initiated by another master on the VMEbus. VMEbus master transfers, initiated by the CPU, cannot be buffered by a cache and what is more may be slow. Therefore to avoid loading the HSPbus and because these accesses are not cached, the CPU modules have a separate VMEbus master interface.

The memory modules also have gained an interface to the VMEbus. This is a slave-only interface which gives access to diagnostic and configuration information on the memory module.

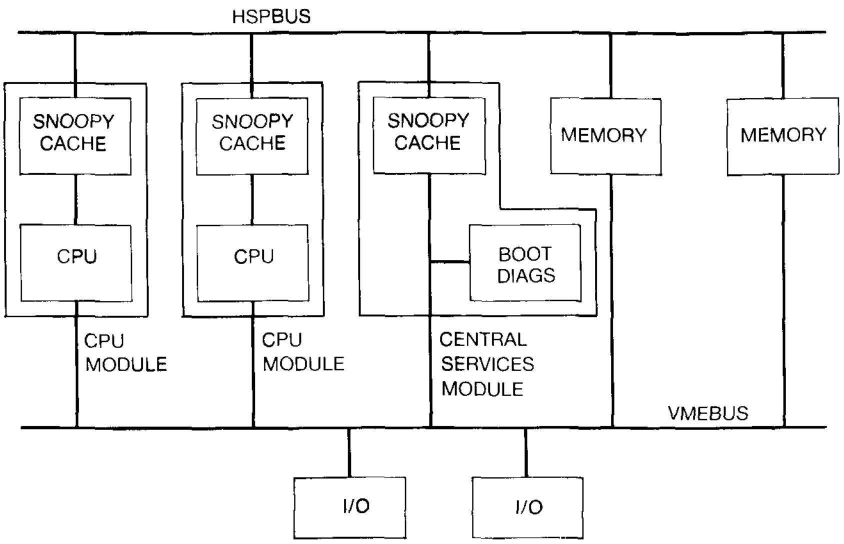


Fig. 2 UNICORN architecture

The following sections describe in more detail each of the major functional modules of the core UNICORN architecture. These are:

- CPU Module
- Memory Module
- Central Services Module

2 CPU Module

The CPU Module is a SPARC based processor module. Functionally it may be divided into several submodules. The submodules to be described in the following sections are as follows:

- Integer and Floating Point Coprocessor
- Cache
- Cache Management and Memory Management Units
- Snoop
- VMEbus Master and Slave Interfaces
- Interrupt Handler.

A block diagram of the CPU Module is given in Fig. 3 which shows the relationship of the various submodules.

2.1 Integer Unit and Floating Point Coprocessor

The CPU module design is targeted at a particular implementation of the SPARC microprocessor; the integer unit runs at 33 MHz and gives a

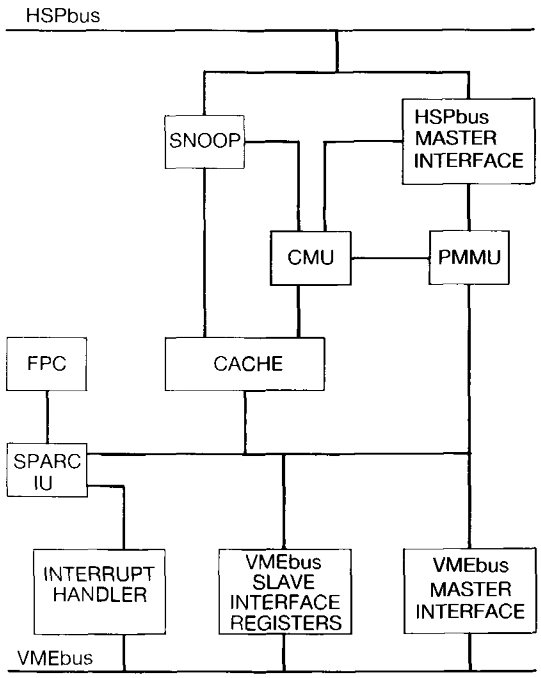


Fig. 3 CPU Module Block Diagram

performance of 15–20 mips. This is described by the vendor as a 20 mips part. The processor has associated with it a floating point chip which interfaces to the main integer unit.

The integer unit does not include memory management and so generates a 32 bit virtual address. This is extended by a 16 bit context number to distinguish one task from another.

Context 0 is the kernel context used by the operating system.

Contexts 1 through FFFE (hexadecimal) are used for user contexts.

Context FFFF (hexadecimal) is reserved for cache flushing (clearing out the contents of the cache).

2.2 Cache

The SPARC CPU generates virtual addresses and it is not possible to translate the address through an external Memory Management Unit (MMU) to access a physical cache within the cycle time of 30 ns. A virtual cache is therefore implemented where the cache is accessed directly with virtual addresses.

The cache has the following characteristics:

- 128 Kbyte cache
- 64 bits wide for refill
- 32 byte line
- Copy back
- Cache coherency.

A copy back strategy, where memory is only updated when a line is displaced, reduces bus bandwidth over a write through strategy. Copy back is necessary because of the bus bandwidth limitations in the multi-processor case. Copy back also increases CPU performance. It does, however, increase the complexity of cache coherency. Figure 4 shows a block diagram of the cache.

The cache comprises five RAM arrays.

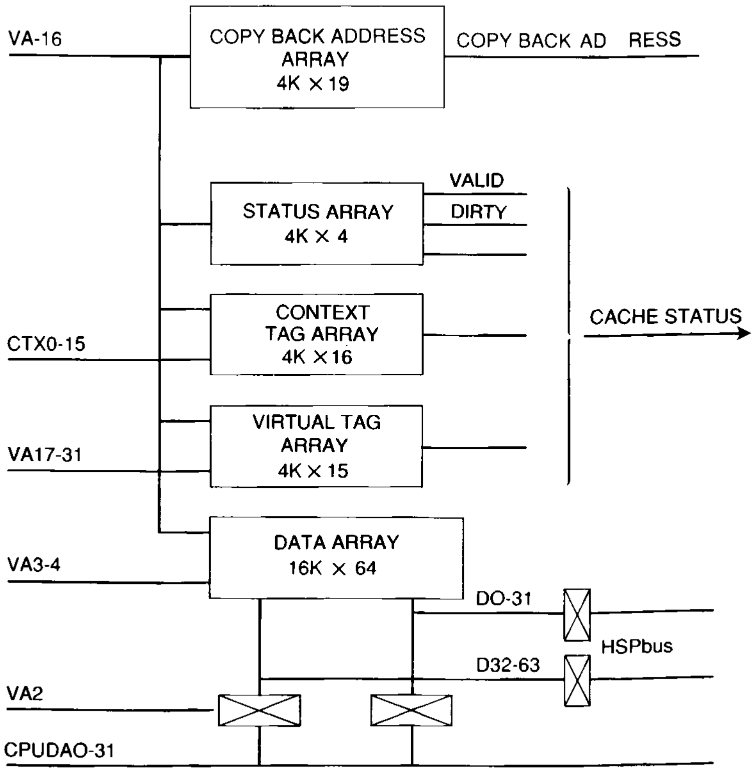


Fig. 4 CPU module virtual cache

The data array contains the data which is cached. It logically has two ports; one to the CPU chip and one to main memory. The performance of the cache is very much dependent upon the product of the miss rate and the time to refill a cache line on a miss. The miss rate is minimised by having a large cache (128 Kbytes), and large line length (32 bytes), and the refill time is minimised by having a wide path to memory, in this case 64 bits wide. The data array organisation is therefore 16K × 64. Address bits VA3–16 address the data array.

When accessing the data array, a cache hit is dependent upon:

- The virtual address tag in the tag array matching the virtual address presented by the CPU chip.
- The context number from the context array matching the context register.
- The status bits read from the status array, e.g. valid.

With a line length of 32 bytes, address bits VA5–VA16 address the tag, context, and status arrays.

Copy back means that writes go only to the cache and not immediately to main memory. Main memory is updated only when a modified cache line is flushed from the cache. Because the memory management unit can only translate addresses for the current context and the supervisor context, it is necessary to save the physical address associated with each line in the cache for use during copy backs. These are saved in the copy back address array.

The cache will need to be flushed under hardware control. On power-on reset the status RAM will be reset to make all cache lines invalid.

The cache will also need to be flushed under software control. This occurs when context numbers are to be reused. As at this point the cache may contain modified lines, it is not possible to flush the cache by resetting the valid bit in the status array otherwise the data in the modified lines will be lost. In this case the context array is reset to FFFF(Hex) and the status bits are maintained. Context FFFF(Hex) is not used by the processor and so effectively the cache has been flushed. Modified lines now in context FFFF(Hex) will be copied out to memory as they are displaced.

2.3 The Cache Coherency Protocol

Support for cache coherency is distributed through the system and resides in the cache controllers, the snoop logic, and the HSPbus protocol. A caching algorithm is defined which gives high performance with full cache coherency. This is a copy back algorithm which requires that each line in the cache can be in one of several states. Four cache line states are required. These are:

- INVALID
- SHARED NON DIRTY (SHARED)
- EXCLUSIVE NON DIRTY (PRIVATE)
- EXCLUSIVE DIRTY (MODIFIED)

The meanings of these terms are:

DIRTY	the cache line has been updated but the corresponding line in main memory has not; therefore this line must not be simply discarded but must first be copied back to main memory
SHARED	the line may be present in more than one cache
PRIVATE	the line is present in one and only one cache.

The status of a cache line will transit from state to state on both processor based activities and bus activity. Bus activity is monitored by the snoop logic described in the following section. The cache consistency protocol is shown diagramatically in Fig. 5 and is described as follows:

Processor activity

Read hit. Data is read directly from the cache and there is no state change.

Read miss. If the line is MODIFIED it is copied back to main memory and the new line is read in. All other caches capture the address of the read and

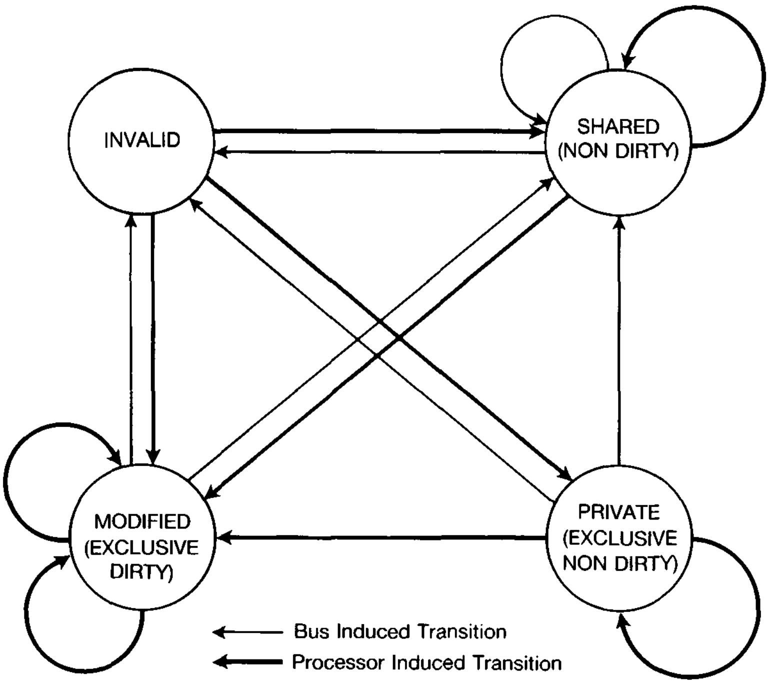

INVALID
SHARED (NON DIRTY)
MODIFIED (EXCLUSIVE DIRTY)
PRIVATE (EXCLUSIVE NON DIRTY)
Bus Induced Transition
Processor Induced Transition

Fig. 5 UNICORN state transition diagram

check whether they have cached that line. If it is in another cache as MODIFIED the read from memory is intervened (intercepted) and the MODIFIED line is written back to memory. The original read then continues. If the line was in another cache as PRIVATE or SHARED, then they will signal the fact via an HSPbus signal and the line will transit to SHARED. The other cache line, if PRIVATE, will also transit to SHARED. If no other cache signalled that it held the line, the line becomes PRIVATE.

Write hit. If the line is PRIVATE, it is necessary to check for write permission before proceeding. If OK the line status changes to MODIFIED. If the state was already MODIFIED, the write proceeds immediately and there is no state change. If the line is SHARED then the physical address is broadcast on the HSPbus (ABI, address broadcast invalidate) to other caches holding the line. The other caches will transit to INVALID and the original cache will transit to MODIFIED.

Write miss. A write must perform a write allocate. This means that the line must be read in before the write can proceed. Therefore first follow the sequence for a read miss, then the sequence for a write hit.

Bus activity

Address Broadcast Invalidate. When an address broadcast invalidate is transmitted over the HSPbus, each snoop unit will look up the address broadcast in its cache. If it hits in the cache then that line will transit to INVALID.

Reads. The addresses of all reads are monitored by the snoop units. If the address hits in another cache and the line status is MODIFIED, then the snoop unit will intervene in the current bus transaction and copy the line back to memory. If it hits in the cache and the line was PRIVATE or SHARED then the snoop unit will signal "shared" on the HSPbus and the status of the line read and the line snooped will both transit to SHARED.

Writes. Line writes will only ever occur when a MODIFIED line is displaced from the cache. By definition a MODIFIED line can only be in one cache and so line writes will not be snooped. The addresses of non-line writes may be snooped in another cache and the status of the snooped line may be PRIVATE, SHARED, or MODIFIED. PRIVATE and SHARED lines will transit to INVALID. A snoop hit on a MODIFIED line will cause intervention and the modified line will be copied back to memory before the original write is allowed to continue.

The cache coherency algorithm may also be described by a state transition diagram. This is given in Fig. 5.

2.4 CPU Module Snoop Unit

The snoop logic will capture all physical addresses which appear on the HSPbus and determine whether the line represented by that physical address has been cached in the CPU's virtual cache.

The snoop therefore contains as tags all of the physical addresses which correspond to the virtual address tags in the CPU's cache.

Using the physical address from the bus as an index into the snoop tag gives a choice of 16 physical tags to compare the physical address with. This is because the page size is 8K and the cache is large enough to contain 16 × 8K pages (128 Kbytes). A hit occurs when a physical address on the bus matches one of the tags. In this case the state of the corresponding line in the CPU's cache must be modified (e.g. invalidated) or the shared state signalled on the bus. As a single physical memory location may map to several virtual addresses (i.e. shared memory synonyms) then there could be several hits in the snoop tag. This would be difficult to handle and so a restriction has been made such that synonyms will displace each other from the cache. There are two possible techniques to achieve this. The first is to align synonyms on 128 Kbyte boundaries, this being the cache size. Synonyms will then naturally displace one another from the cache. An alternative but more complex method is self snoop. Whenever a cache line refill takes place, the rest of the cache is snooped to check for synonyms. If a synonym is found then the entry is invalidated. Of course if it was MODIFIED it is copied back first. Either technique ensures that there is only ever one instance of a line of data in the cache and there are no synonyms.

As already described the snoop is required to match 16 tags simultaneously. In effect the snoop is a 16 way set associative cache. Normally the implementation of such a cache would be impractical in terms of the amount of hardware required. However, there is now available a 4 way associative tag RAM as a single chip. This makes a 16 way associative tag feasible with four chips.

There are two operations required of the snoop. The operation already mentioned is the snoop itself. The other operation is the loading of the snoop tag with the physical addresses corresponding to the virtual addresses in the CPU's cache, i.e. on a cache refill.

On a cache refill the snoop array is addressed by the virtual address from the processor. VA0–VA4 address the byte within the cache line and so are not used here. VA5–VA12 are used to index into the tag selecting one of 256 sets, each set containing 16 tags. VA13–VA16 are used to select the way where the physical address used in the refill will be loaded as a new tag using PA13–PA31.

On a bus snoop the physical address from the bus is captured in a register. This is now used to address the snoop array. PA0–PA4 again are not used. PA5–PA12 are used to index into the array to select a set containing 16 tags. PA13–PA31 is then compared associatively with all 16 tags.

There are 16 separate hit lines from each of the ways but, because of the restriction on synonyms, only one hit line will assert if there is a hit. A 16 to 4

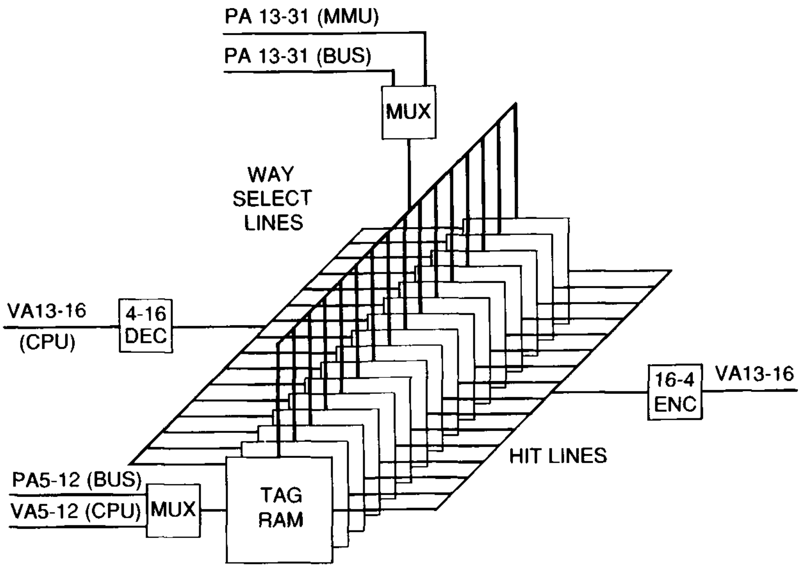


Fig. 6 CPU module bus snoop block diagram

encoder is then used to recreate the virtual address bits VA13–VA16 which together with PA5–PA12 give the location of the line in the CPU's cache which has to be modified. With a page size of 8K, PA5–PA12 are the same as VA5–VA12, i.e. they are not translated. Figure 6 shows a diagram of the snoop.

2.5 Cache Management and Memory Management Units

The purpose of these submodules is to translate the virtual address emitted by the integer unit and to process integer unit cache misses and service snoop hits.

The memory management function provides a translation buffer of 64 page table entries and has hardware controlled page table walking for replacing entries on a miss.

The Cache Management Unit is implemented as a microprogrammed state machine executing the cache coherency algorithm.

2.6 VMBEbus Master and Slave Interfaces

The VMEbus Master function is required when the CPU needs to access devices (slaves) on the VMEbus. An ASI (Address Space Indication) decode is defined for addressing the VMEbus. On decode of this ASI the address is presented directly to the VMEbus. Both the cache and the MMU are bypassed. The VMEbus is arbitrated for and the access to the slave takes place, either sourcing data from the CPU for a write or returning data to the CPU for a read.

All VMEbus accesses are qualified by an address modifier. The address modifier presented on the VMEbus is given by the contents of an internal register. Some of the address modifier codes define the following types of access.

- 32 bit (extended) addressing
- 24 bit (standard) addressing
- 16 bit (short) addressing.

A VMEbus slave module is also required. This will provide a number of registers in a user-defined VMEbus A16 address space. This reserved space is known as the Controller Interface Table (CIT). The address of the base of these registers will be determined by the geographical address lines picked up from the backplane, i.e. the slot number. Registers may be read from the VMEbus to provide the following information.

- Board type (CPU)
- Board status (fail, etc.)
- Interrupt status.

Registers may be written to from the VMEbus to drive the following on board functions.

- Board reset
- CPU halt and restart
- to cause and mask interrupts.

2.7 Interrupt Handler

An interrupt handler is provided to handle all seven VMEbus interrupts. In a multiprocessor system all CPUs will have the VMEbus interrupt handler logic but the task of interrupt handling at any one level will only be handled by one CPU at a time. The VMEbus interrupts are combined with other interrupts for presentation to the integer unit.

3 Memory Module

The memory module resides in the HSPbus and provides 32 MBytes of Memory using 1 MBit DRAMs or 128 MBytes using 4 MBit DRAMs. The boards are capable of being depopulated to 16 MBytes and 64/32 MBytes respectively. Figure 7 shows a block diagram of the memory modules.

The memory module receives a 32 bit address and sources or sinks data on a 64 bit data bus. The memory is designed to operate in either 32 bit or 64 bit

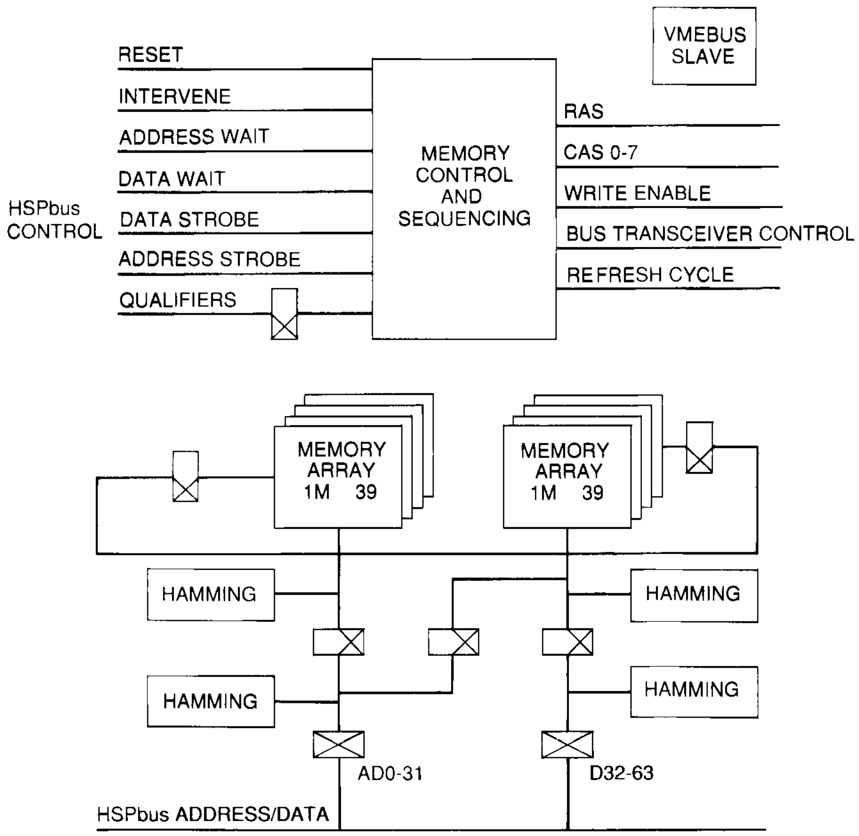


Fig. 7 UNICORN Memory Module

mode. In 32 bit mode, the data lines used are AD0–AD31. In other words the data is shifted to the least significant end of the data bus.

The memory module is capable of handling the following functions in 64 bit mode:

- Read of 64 bits, representing 8 consecutive bytes.
- Read of 2 × 64 bits, representing 16 consecutive bytes.
- Read of 4 × 64 bits representing 32 consecutive bytes.
- Read of 8 × 64 bits representing 64 consecutive bytes.
- Write of 64 bits, representing 8 consecutive bytes.
- Write of 2 × 64 bits, representing 16 consecutive bytes.
- Write of 4 × 64 bits representing 32 consecutive bytes.
- Write of 8 × 64 bits representing 64 consecutive bytes.

With writes for each 64 bit quantity 8 validity bits are presented, with the data corresponding to the bytes which are to be actually written.

The following operation is supported in 32 bit mode:

- Read of 32 bits, representing 4 consecutive bytes.
- Write of 32 bits, representing 4 consecutive bytes.

With writes for each 32 bit quantity 4 validity bits are presented, with the data corresponding to the bytes which are to be actually written.

Reads and non-line writes are capable of being aborted by the "intervene" signal on the HSPbus. This turns the transaction into a line write and data is presented to the memory from a cache other than the one which was doing the original transaction. Upon completion of the write, the original transaction continues. This feature is required for the cache coherency protocol and is known as intervention.

Memory protection is provided by Hamming checks. Writes to the memory generate Hamming check bits which are written with the data. Hamming is generated over 32 bits and so each 32 bit word has associated with it 7 Hamming bits. Reads cause the Hamming bits associated with the data to be checked. An error will cause the HSPbus signal MERR* (Memory ERRor) to be asserted. Because of the fast access time of the memory module it is not possible to correct Hamming errors on the fly. On detection of an error, therefore, the memory module performs an internal correction cycle after the event. The master initiating the transfer has to repeat the failing transaction. The retry will succeed if the error was soft and was corrected internally by the memory. If the error was hard but correctable, it is necessary to set the memory into on the fly correction mode. This slows down the access time of the memory module but allows hard correctable errors to be corrected.

Although the main interface to the memory is via the HSPbus, a VMEbus slave interface is also required for status and configuration. This will provide a number of registers in the user defined VMEbus A16 address space. These registers are used for a Controller Interface Table (CIT). The address of the base of these registers is determined by the geographical address lines picked up from the backplane, i.e. the slot number. Registers may be read from the VMEbus to provide the following information.

- Board type (Memory, size)
- Board status (Hamming failure)

Registers may be written to from the VMEbus to drive the following on board functions.

- Message register
- Memory base address
- Memory bank enable.

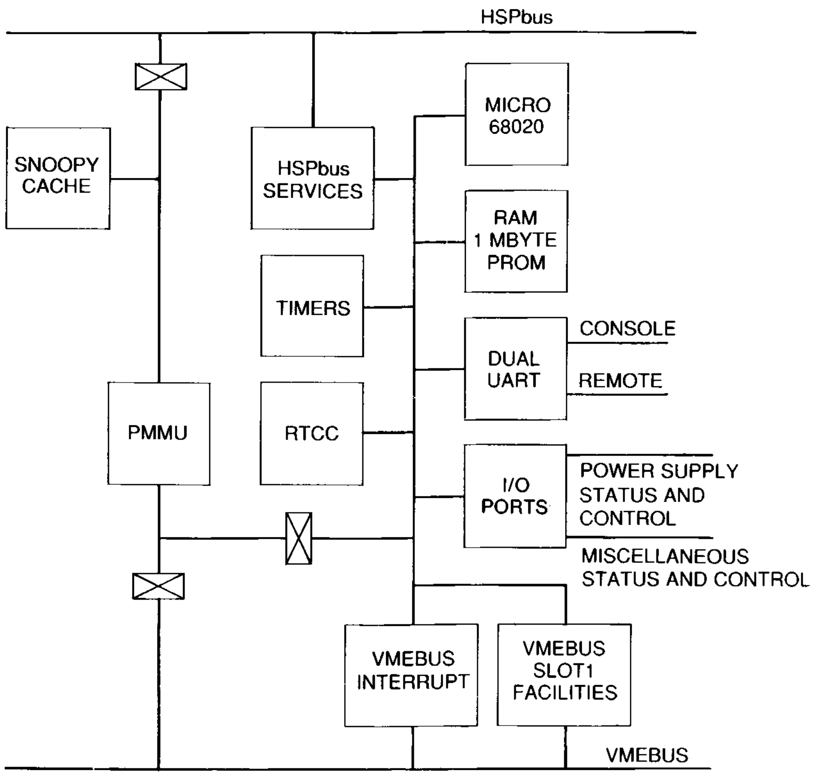


Fig. 8 Central Services Module

4 Central Services Module

The Central Services Module, as its name implies, provides services for the system. Only one CSM is configured into a system. It provides the following functionality:

- HSPbus services.
- VMEbus Slot 1 controller functions.
- VMEbus slave cache interface to memory.
- Boot and diagnostic facilities.

Figure 8 shows a block diagram of the CSM.

4.1 HSPbus Services

This functional block provides the following facilities for the HSPbus.

- Clock – this is a free running 1:1 mark/space 60 ns clock which is the bus clock from which all bus transitions are timed.

- Reset – this is the reset on the HSPB and is controlled by a bit in a register which is controlled from the diagnostic micro.
- HSPbus Error – this is a signal which is asserted if the bus had been owned for longer than 15 microseconds by a single module.

4.2 VMEbus Slot 1 Controller Function

The VMEbus System Controller functions listed below are provided:

- System Clock driver
- Serial Clock driver
- SYSRESET driver by a bit in a register controlled by the diagnostic micro
- Arbiter Module
- Bus timer module
- IACK (Interrupt ACKnowledge) daisy chain driver.

4.3 VMEbus Slave Cache Interface to Memory

This functional block provides a cache for transfers initiated by the VMEbus. Unlike the CPU caches, the VMEbus cache is a physical cache. However, it does conform to the cache coherency rules established for the system. For efficiency and maximum bandwidth, block mode transfer is used on the VMEbus where possible. To avoid the VMEbus I/O controller having to take notice of page boundaries, an MMU is included between the VMEbus and the I/O cache. I/Os are then allocated segments which link into a process's page tables.

The operation of the I/O cache is as follows:

Reads All reads will cause a line to be allocated in the cache and subsequent reads within that line will come directly from the cache.

Writes Writes follow a rather more complicated procedure dependent upon whether the write is a block transfer and where the transfer begins and ends relative to line boundaries.

The first write of a non block mode write will cause write allocation, i.e. the relevant line will be read into the cache. If the line is SHARED, an address broadcast invalidate will be sent over the HSPbus to establish the line as PRIVATE and the line will be modified. Subsequent writes to that line take place without HSPbus action.

For a block mode write, if the write starts not on a line boundary, then a line in the cache is write allocated and the action is as in the previous paragraph.

If the block mode write begins on a line boundary then the write takes place to the cache line. The previous contents of the line are flushed to memory if the line status was MODIFIED. Subsequent writes also go to the cache. If the block mode write continues to the end of the line then an address broadcast invalidate is sent over the HSPbus and the line status is set to MODIFIED. Up to that point, the line status was set to a special reserved value indicating this condition. An address broadcast invalidate is sent because the line may be SHARED.

If the block mode write ends before the end of the line, the remainder of the line is read in from memory. If the line was read in as SHARED, an address broadcast invalidate is sent over the HSPbus before the line state is then set to MODIFIED.

Snooping The I/O cache must maintain cache coherency with all other caches in the system and so snooping is required. Note that being a physical cache, snooping uses the physical address from the bus directly as a cache index.

4.4 Cache Configuration

The I/O cache size is 64 Kbytes containing 2K lines each of 32 bytes. As the addressing characteristics of I/O are different from that of a CPU, the address bits used for accessing the cache are ordered differently. Address bits used for indexing the cache are PA0–4 and PA13–23. Address bits PA5–12 and PA24–31 are used as tags. This arrangement has the characteristics that an I/O in a single page will use only one cache line.

4.5 Boot and Diagnostic facilities

These are provided by a microprocessor controller subsystem resident on the CSM. As this will be involved in power-on diagnostics, power supply sequencing and remote access, this particular subsystem will be powered separately from the rest of the system. The following functional units are provided in the Boot and Diagnostics subsystem:

- Microprocessor
- RAM
- PROM
- Local console interface
- Remote console interface
- Real time clock, calendar (battery backed)
- Timers
- VMEbus interrupter.

Power-on sequencing will be performed by the Boot and Diagnostics subsystem upon command from the control panel (on/off switch) or the remote console (remote access). It will check that all power rails are in specification before moving to the next step.

The core configuration is then established by groping the system, given that the CFUs and memories have configuration information accessible by slot position over the VMEbus.

Next the Central Services Module is tested, followed by the memory modules. A CPU test program will then be downloaded into memory and a CPU released to execute it. In a multi-CPU system, the CPUs are tested one at a time.

Once all basic boards have checked out OK, a boot sequence is downloaded to memory and control is passed to a CPU.

Note that because of the control that the Boot and Diagnostics part of the CSM has over the system, it is not necessary for the CPUs to have boot PROMs.

5 I/O Controllers

I/O for the UNICORN system is provided by I/O controllers which sit on the VMEbus. These are standard VMEbus cards.

5.1 VMEbus Microlan II Interface

Microlan II is an ICL proprietary two wire multi-dropped interface which is used to connect Microlan II based terminals and workstations.

5.2 VMEbus Communications Controllers

Two VMEbus communications controllers are available.

One provides four independent full duplex V24 communications channels which operate at speeds up to 19·2 Kbps.

The other provides two independent channels operating at speeds up to 64 Kbps.

5.3 VMEbus Ethernet Controller

This is a single channel controller which provides UNICORN with the facilities to support OSLAN 100, 200, 300, and 500.

5.4 VMEbus SCSI Controller

This is a two channel controller which provides the ability to connect discs (magnetic and optical), and tapes (and, potentially, high speed printers) via the industry standard SCSI interface.

5.5 VMEbus Asynchronous Communications Controller

This is a 16 channel controller which provides the ability to connect VDUs, printers, and modems via the industry standard RS232 (CCITT V24) interface at speeds up to 38·4 Kbps.

5.6 VMEbus Repeater

The VMEbus repeater provides the means to expand the VMEbus beyond the 20 slots provided in the main cabinet card cage. The repeater comprises two VMEbus cards which are linked together by a cable. One resides in the main card cage and the other sits in the card cage in an expansion cabinet.

6 Summary

UNICORN provides a very flexible architecture to support its role as a multi-user UNIX platform.

The hardware allows great flexibility in configuration allowing cost effective solutions to be realised from 32 users to several hundred users.

The high performance of the SPARC microprocessor is supported by high performance caches with total hardware cache coherency through the system. This both improves performance and relieves the operating system software of the burden of maintaining cache coherency.

DRS6000 (UNICORN) software: an overview

T.M. Cole
Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berkshire

Abstract

This paper gives a very brief overview of the version of UNIX which has been ported to the UNICORN. It also describes, in more detail, the extensions which have been made for UNICORN. In particular, the "extended streams" facility is described. This provides an environment which can be ported relatively easily to I/O Controller boards (IOCs) which can then support standard UNIX streams modules. This gives much greater flexibility in system configuration, as some facilities (embodied in streams modules) can be relegated to IOCs at a very low implementation cost.

Introduction

The UNICORN will support AT&T's[1] UNIX[2] System V Version 4 in its SPARC[3] ABI (Application Binary Interface) form – see [Ref. 1, 2, 3]. UNIX System V Version 4 incorporates the facilities of UNIX System V Version 3.2 and the Microsoft XENIX[4] system, and some of the facilities of the Berkeley BSD[5] system and Sun Microsystems' SunOS[6]. This will conform to the POSIX[7] (Portable Operating System Interface for Computer Environments) and X/OPEN[8] standards – see [Ref. 4, 5]. The generic source is supplied to us by AT&T, in a form as implemented on an AT&T 3B2 series computer, and we "port" it to the UNICORN and merge in ICL's "value added" components (e.g. Microlan support). This paper describes the software as it applies to this port. A general description of the software architecture of UNIX is available in [Ref. 6], for instance.

The "Porting" Task

The porting task involves taking the generic source and reworking the machine-specific parts to conform to the target machine – in this case the UNICORN. This includes removing all the AT&T 3B2-unique programs and interfaces, and modifying or rewriting others – for example the cartridge tape programs – where UNICORN equivalent hardware exists. The largest part of this task concerns the peripheral access, although a substantial amount of effort also has to be devoted to memory management and the bootstrap process.

The UNICORN hardware is described briefly in an accompanying paper in this issue. [Ref. 7]. There are three classes of hardware component in the UNICORN which are capable of executing code. The CPU (Central Processing Unit) is the only one concerned with executing application programs, as well as being responsible for the kernel functions: it does not have direct access to peripheral devices. The CSM (Central Services Module) and IOCs (Input/Output Controllers) are the only components with direct access to peripheral devices. The CSM is responsible for the system console device and an auxiliary RS232 port intended for use for diagnostic purposes. The CSM is also responsible for starting the system and for monitoring the health of the system, for example detecting cabinet overtemperature. The IOCs are responsible for all the other peripheral devices. Only the CSM and CPU have direct access to main memory: the IOCs are required to access main memory via the CSM.

The division of responsibility among the computational units requires coordination. This is achieved by exchanging messages over a bus which is accessible to all. This bus is called the IObus and is an implementation of the standard VMEbus [Ref. 8]: the "VME" here may stand for "Versa Module European", in which case the "Versa" bit refers to a Motorola proprietary bus – but there are various opinions about this.

Peripherals are accessed by the UNIX kernel via device drivers. There is a standard interface between the drivers and the kernel known as the DKI, Driver-Kernel Interface [Ref. 9]. There are three types of device driver, known respectively as **block** (mainly for disc access), **character** (also known as **raw** or **physical** access when applied to disc or tape), and **streams** [Ref. 10]. Of these, by far the largest variety of usage goes to the streams drivers, as these support the communications facilities which are central to UNICORN's departmental systems role.

Streams were introduced in UNIX System V Version 3 kernel to enable communications protocols to be modularised. [Ref. 11] defines as follows:

"**STREAMS** A set of kernel mechanisms that support the development of network services and data communications **drivers**. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures."

The OSI seven layer model, for example, lends itself to modularisation, the various layers providing the initial basis for defining the functions and interfaces of the component modules. The interfaces between the modules are expressed in a standardised (streams) message passing form. This standard enables them to be interconnected in any rational sequence at run time, without the need for further recompilation or linking. There is a "streams" scheduler in the kernel which causes the various streams modules to be executed from time to time as required, to process their input messages and modify them or perhaps generate new messages which are output to the next module in the stream.

The UNICORN extends this streams environment to include the CSM and IOCs so that communications modules may be IOC, CSM or CPU resident without needing to rework the modules themselves, or any abridging of the former flexibility of interconnection. If versions of the modules are available ready-linked for the IOC, CSM or CPU environments, the choice of stream module location can be left to run time. This extended environment is referred to as "Extended Streams" in the context of the UNICORN port.

The porting tasks associated with the non streams drivers are fairly standard, and will not be discussed here.

Extended Streams on the UNICORN

This scheme is realised with:

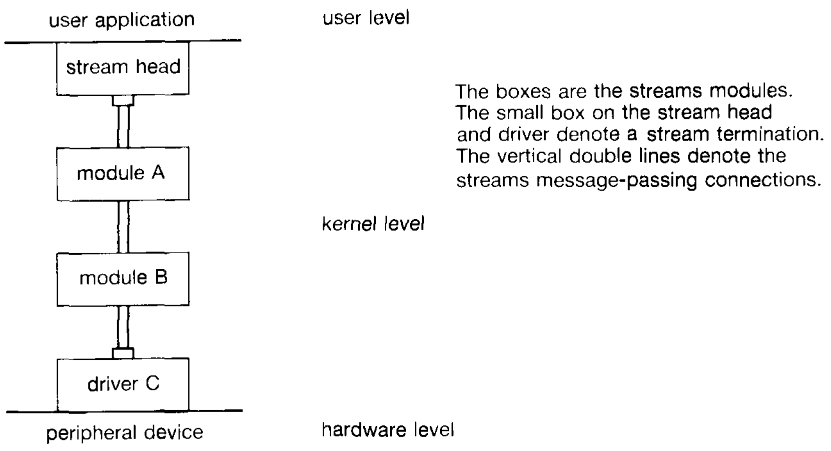
- a driver which can pass messages along the IObus
- a miniature kernel environment on each IOC sufficient to supply all the needs of streams modules (qv DKI [Ref. 9])
- a set of "stub" modules to represent, in the kernel environment in main memory, those modules absent from that environment.

The "stub" modules are known collectively as **relays**, as they transparently "relay" messages across the IObus. The miniature kernel environment on each IOC is known as the **ISMkernel** (Intelligent SubModule **kernel**).

UNIX streams are described in [Ref. 10], but the following very brief and simplified overview may give a sufficient background against which to present the idea of Extended Streams.

A stream consists of a stream head (which mediates between the kernel environment and the user application); optionally followed by some streams modules; and terminated by a streams driver. The stream is constructed when the driver is first opened by the user application: this links the driver to the stream head and then proceeds to "push" the modules onto the stream immediately below the stream head (see Fig. 1).

The UNICORN Extended Streams could implement this stream where, say, module B and the driver C were IOC-resident, by substituting relays for B and C in the kernel resident stream (see Fig. 2). The relays on the CPU pass any messages they receive to the IOC relay, and pass upstream any messages they receive from the IOC relay. The link from relay C remains dormant until B is "popped" from the stream, as messages from the IOC are passed straight to relay B, and relay B receives any messages coming downstream before relay C.


user application
user level
stream head
The boxes are the streams modules. The small box on the stream head and driver denote a stream termination. The vertical double lines denote the streams message-passing connections.
module A
kernel level
module B
driver C
peripheral device
hardware level
Fig. 1

Our goal was to avoid making any changes to the generic UNIX code supplied to us. The use of relays achieves this because the generic UNIX code can never detect the substitution. Each IOC-resident module or driver has to supply a corresponding relay which is able to locate "its" IOC and pass the IOC address to the relay support code. This structure could be advantageous as it is generally true that messages are more frequent at the lower levels of the communications protocol stacks. Hence the lower levels may be devolved to multiple instances of particular types of IOC.

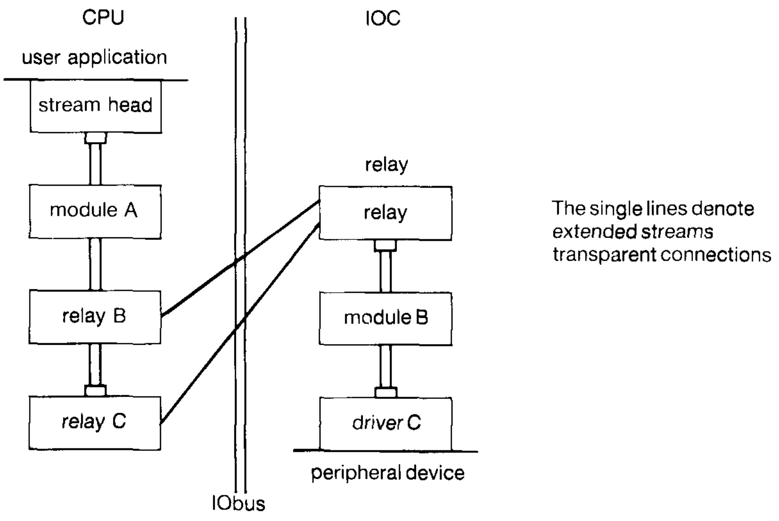


Fig. 2

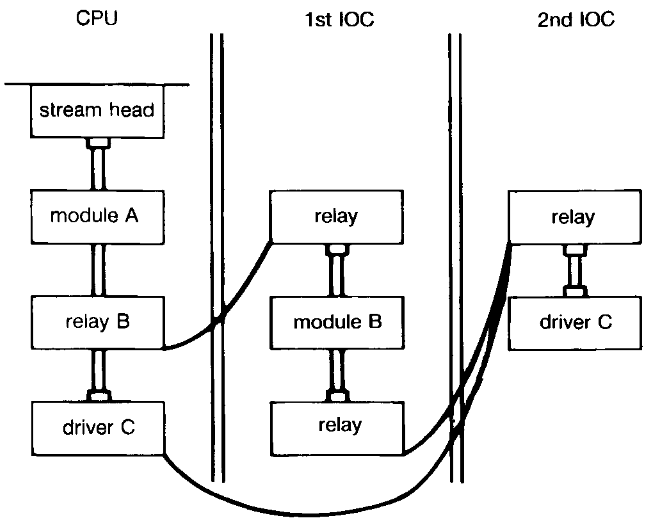


Fig. 3

In the UNICORN, IOCs can communicate with each other on the IObus independently of the CPU. This allows a stream to be extended across more than one IOC (Fig. 3). This is not very useful in its own right, but can be exploited when streams multiplexors are involved (see below).

Some streams drivers are used to multiplex or demultiplex other streams connections. In this case, a single "upper" stream terminates on reaching the multiplexor, which acts as the origin for one or more streams on the "lower" side. This is constructed at run time out of a collection of streams which have already been opened and set up (Fig. 4), one of which is the stream to the multiplexor. The streams are linked to the multiplexor, one by one, until the structure is complete (Fig. 5). The stream heads which belonged to the "lower" streams are now redundant, and would be freed by the application. If modules A1 and A2 were IOC-resident, a multiplexor could be used to "drive" more than one IOC (see Fig. 6). The relay on the first IOC which was used to link relay A1 to the module A1 has become redundant in a similar way to the way the stream head would have become redundant had the stream been CPU-resident.

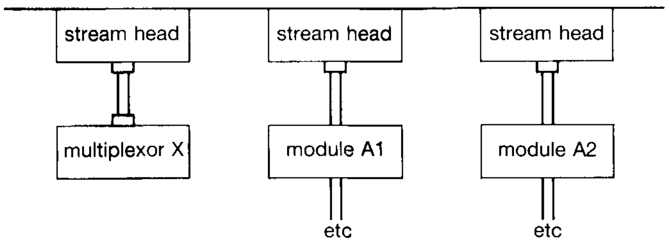


Fig. 4

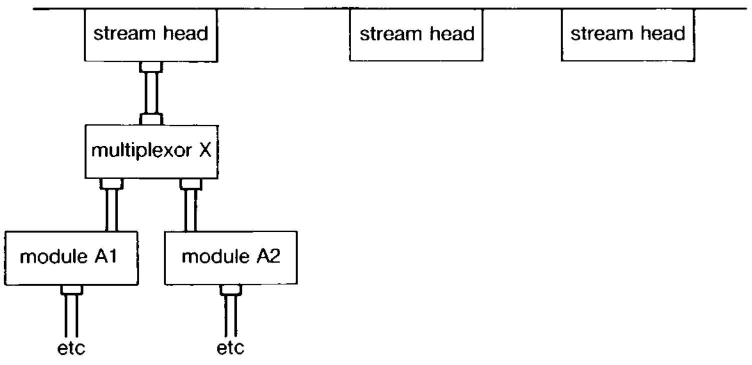


Fig. 5

However, this logical structure could also be built up as follows, with the multiplexor itself placed on one of the IOCs (see Fig. 7). In this case, the links from relays A1 and A2 to their respective IOCs become dormant until the multiplexor is dismantled, as the relay X intercepts all messages coming downstream and dispatches them to the IOC relay. Similarly, all messages from modules A1 and A2 flow to the multiplexor on the first IOC, and then via the multiplexor relay. This form may be advantageous if the multiplexor is capable of receiving messages from, say, module A1 and sending them on to module A2. Such a flow would never directly concern the CPU in this configuration.

The extended streams code is structured into layers to facilitate porting to other systems and IOCs, and also to allow for non-streams usage (see Fig. 8). There are various sub-systems which exchange messages, and are classed as different "services" within this layered structure. Hence, the extended streams use becomes the "extended streams service". Some functions required by the ISMkernel environment (e.g. passing messages to be printed on the system console) are provided by a "kernel utilities" service.

At the first release, the Microlan IOC and CSM will be fully implemented to

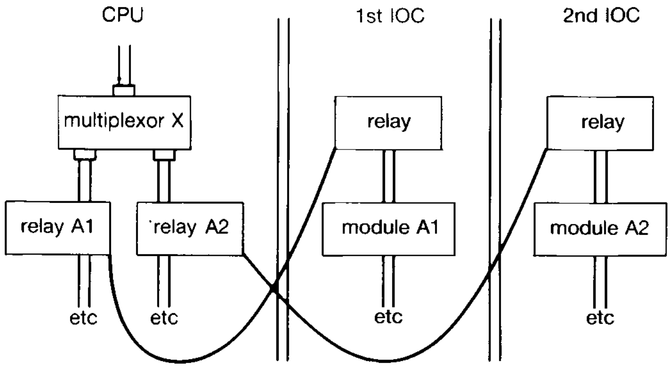


Fig. 6

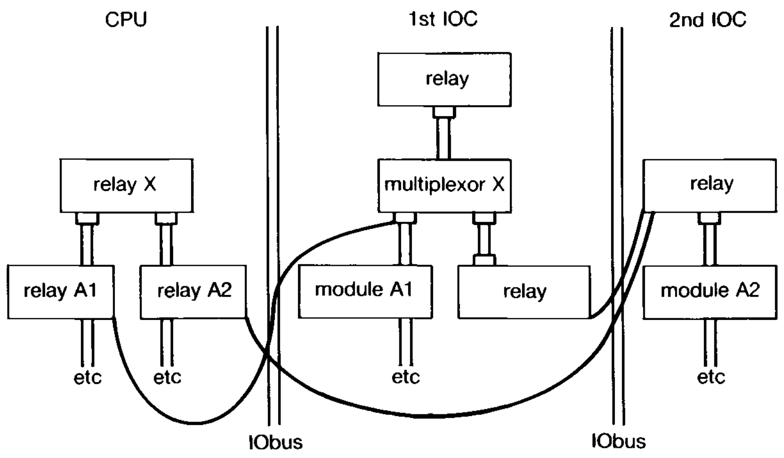


Fig. 7

this plan. The X25 IOC will interface some already implemented non-streamed software to the IObus message passing interface so that the CPU-resident parts of the protocol can be implemented as standard streams modules. In the future, it is intended to implement a high performance Ethernet[9] IOC to this scheme. Once established, the only rework required to introduce a new IOC should be to reimplement the IOC-dependent part of the ISMkernel.

Other aspects of the UNICORN software port

All access to main memory from the CSM or IOCs is made via a virtual-to-

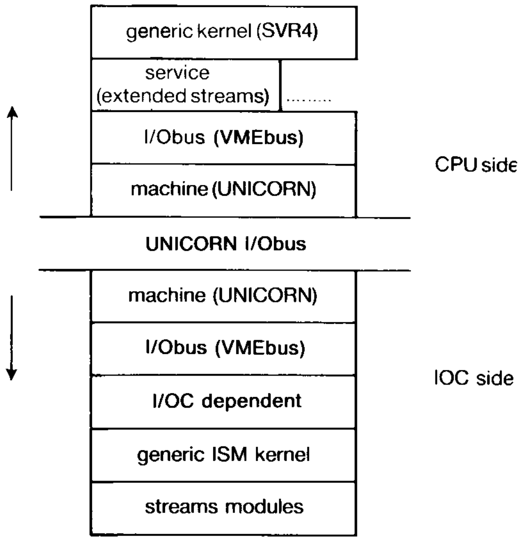


Fig. 8

physical address translation on the CSM board. This relieves the IOCs and CSM from having to "know" anything about the UNICORN memory management. It also removes the need for the CSM and IOCs to support "scatter/gather" DMA. The addresses passed to IOCs are virtual addresses within an address space which is reserved for i/o. Buffers are mapped into this space as required. Kernel-resident buffers are permanently mapped in. Application program-resident buffers are mapped in as required. The first 16 Mbytes of this i/o address space are reserved for buffers which are to be accessed by IOCs with only 24-bit addressing capability. These buffers can nevertheless exist anywhere within the HSPbus physical main memory (128 Mbytes maximum at first release).

The CSM is used to control the bootstrap process, including running establishment checks on the CPU and memory boards. In this way, it is not necessary to include any PROM code on the CPU boards. The CSM can also be used to run ETS (*E*ngineering *T*est *S*oftware) which can be controlled remotely if required.

Conclusion

The Extended Streams facility will allow more flexibility in configuring UNICORN systems, especially where large amounts of communications are to be supported. The modules, multiplexors and drivers which implement the communications protocols may be positioned in IOCs or in the CPU to optimise the system performance. Future IOCs and communications facilities may be introduced with a minimum of rework, without detracting from this flexibility of configuration.

End Notes

[1] AT&T is a registered trademark of AT&T in the USA and other countries
[2] UNIX is a registered trademark of AT&T in the USA and other countries
[3] SPARC is a trademark of Sun Microsystems Inc.
[4] XENIX is a registered trademark of Microsoft Inc.
[5] BSD is a trademark of the University of California at Berkeley
[6] SunOS is a trademark of Sun Microsystems Inc.
[7] POSIX is a trademark of the Institute of Electrical and Electronic Engineers Inc.
[8] X/OPEN is a trademark of the X/OPEN Company Ltd.
[9] Ethernet is a trademark of Xerox Corporation

References

1 'AT&T System V Interface Definition', Issue 3 – SVID89
2 'AT&T System V Application binary Interface'
3 'AT&T SPARC Processor Supplement'
4 IEEE standard 1000.3 – 1988
5 'X/OPEN portability guide', Issue 3
6 BACH, M.J.: 'The Design of the UNIX Operating System'. Prentice Hall, ISBN 0-13-201799-7
7 POSKITT, G.: 'The UNICORN Architecture'. ICL Tech. J. 6, 4 November 1989
8 IEEE standard 1014: 'VMEbus Specification Revision C1 Oct 1985'. PRINTEX Publishing Inc.
9 'AT&T System V Device Driver Interface/Driver-Kernel Interface'
10 'AT&T System V Streams Primer'
11 'AT&T System V Programmer's Reference Manual'

Electromechanical Design of DRS6000 (UNICORN)

Roger Pullen
Advanced Servers Product Centre, ICL Office Systems, Bracknell, Berks.

Abstract

The paper gives the basic principles underlying the physical design of UNICORN and short accounts of the cabinet design and construction and of the equipment modules built into it: logic rack, disc units, cooling system, power supply and internal interconnections. It gives also the national and international standards to which UNICORN conforms.

1 Basic Principles and Strategy

The electro-mechanical development process for UNICORN has required a wide range of activities and called upon many diverse skills to produce the end product. From the designers point of view, the goal was to produce a simple cost-effective solution to a set of defined requirements. For UNICORN, these were:

- Small floor-standing, desk-high cabinet
- Able to operate in an office environment
- Aesthetic styling – part of a family of office products
- Single design to cover all international requirements
- Must meet all statutory safety requirements worldwide
- Must be able to use existing production line facilities and more specifically the Ashton Mercury production line
- Easy to install
- Minimum time to repair and service
- Easy to re-configure.

The strategy we adopted in setting out to meet these requirements was:

- To design a simple unit to cover a wide range of system configurations and avoid costly cabinet swap-outs in the field
- To consist of simple fabricated frame and cover construction
- To design for minimum number of mechanical fixings
- To create a flexible design catering for either inhouse, subcontract or overseas manufacture
- To house industry-standard units, i.e. 19″ rack type peripherals
- To create flexible cabling system and eliminate connector bulkheads

- To maximise the use of computer aided tools throughout the design process
- To use a common power supply throughout the product range to reduce spares holding. This should also cater for international mains voltages and frequencies
- To design in quality through continuous validation throughout development
- To offer an uninterruptable power supply interface
- To apply Value Engineering techniques throughout development.

Implementation

Past experience in the design of similar products has highlighted the advantages of using a simple open frame construction to house bolt-in equipment. This type of design enables future upgrades to be incorporated with the minimum of alteration to the basic cabinet. UNICORN has followed this principal throughout.

The UNICORN cabinet has been designed to provide a robust construction, with a pleasing appearance that blends agreeably with most environments, and offers a family image with other ICL products. The simple, but ingenious, construction gives a wide choice of mounting positions for individual bolt-in modules or alternatively standard 19″ rack mounted equipment. Additional cabinets can be abutted to expand the system. The need to obtain relevant company and international approvals/certification has also been considered. Figure 1 shows the basic construction and the major items of equipment.

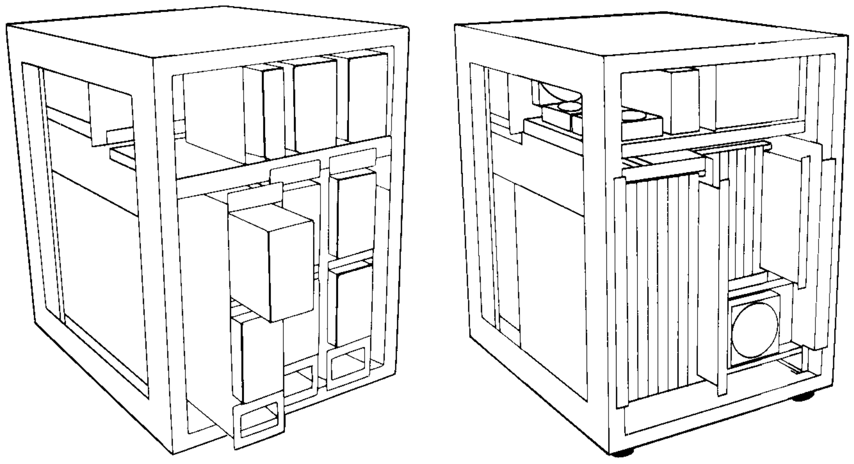


Fig. 1

2 The UNICORN Cabinet

2.1 Cabinet construction

Frame The cabinet frame is constructed of six pre-plated steel (Zintec) members – base, top and uprights spot-welded together to form a simple, robust, open framework which provides:

- 19″ compatible front and rear access
- Four adjustable feet to level the cabinet on uneven floors
- Four castors for mobility during installation
- Versatile design for mounting of equipment
- Flexible system for securing and earthing interface cables without the restriction of bulkhead connectors.

The frame has no costly secondary finishing operation as it is totally obscured by the decorative (external) covers.

External covers The appearance of the cabinet is designed to conform to the ICL product styling requirements to ensure visual compatibility with other ICL equipment, e.g. workstations.

The covers have been designed to enable them to be safely stored until the cabinet has been fully assembled and tested, i.e. covers are fitted just prior to packaging the equipment for delivery.

The painted steel top and side covers are secured in position by integral metal clips which engage in rectangular cut-outs on the frame. This has two advantages; firstly, there is no visual retention method, and secondly, the metal clips provide excellent earthing continuity for safety and radio frequency interference (RFI) shielding.

Front and rear door panels are moulded in Noryl structural foam. This material is a standard engineering plastic with a foaming agent added during the moulding process, which produces a fine, rigid honeycomb construction with a good impact strength. The cooling vents are an integral part of the moulding on both panels, but two secondary mouldings are added to provide access flaps to the exchangeable peripheral devices and the panel securing lock. See Fig. 2.

2.2 Cabinet Equipment Modules

There are five basic modules:

- Logic rack
- Disc modules
- Power supply shelf
- Fan shelf
- Control Panel

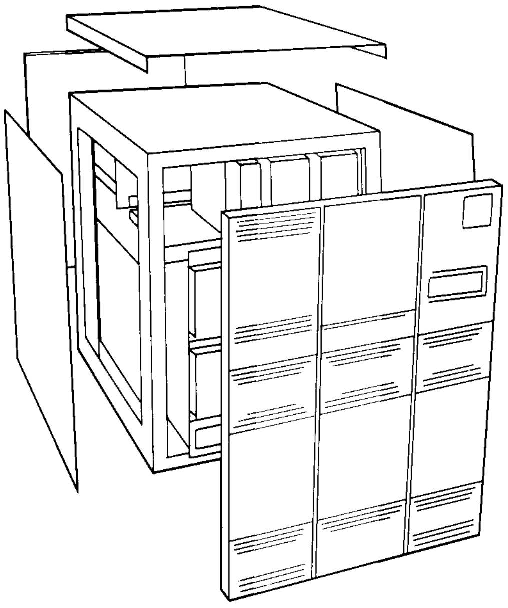


Fig. 2

Except for the logic rack, all modules are easily accessed from either the front or rear of the machine.

All of the modules except the logic rack are secured by either two or four fixings, with electrical connections plugable for ease of fitting and removal, i.e. for servicing.

2.2.1 Logic rack: The logic rack has been designed to support the industry-standard VME racking specification, and accepts standard 366·7 mm × 280 mm (9U) 9 off and 233·35 mm × 160 mm (6U) 11 off printed circuit boards plugged into a common backplane. The different depths of PCBs resulted in ICL-designed metalwork to enable advantage to be taken of VME standard proprietary parts – extrusions, tapped strips, insulators, guides and fixings. See Fig. 3.

A unique feature of the logic rack is the provision to allow small (I/O) PCBs (366·7 mm × 100 mm) to be mounted directly behind the existing PCBs; these mate with reverse DIN connectors on the backplane. Extensions to the rack side panels allow additional extrusions to be mounted both top and bottom, which in turn form the mountings for the card guides for the (I/O) PCBs.

The 6U PCB area of the rack accepts either ICL-designed or externally-sourced PCBs, all fitted with standard VME front panels.

The 9U PCBs and front panels are totally ICL-designed. Particular attention

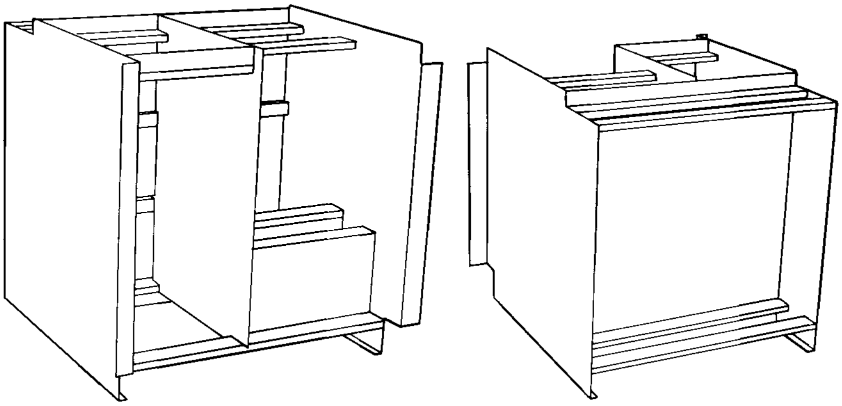


Fig. 3

has been given to the design of the 9U PCB front panels so that a good EMC seal is achieved between the rack side panel and each PCB front panel. The 9U PCBs are also fitted with a purpose-designed board stiffener to assist with the alignment of the three connectors fitted to the backplane. See Fig. 4.

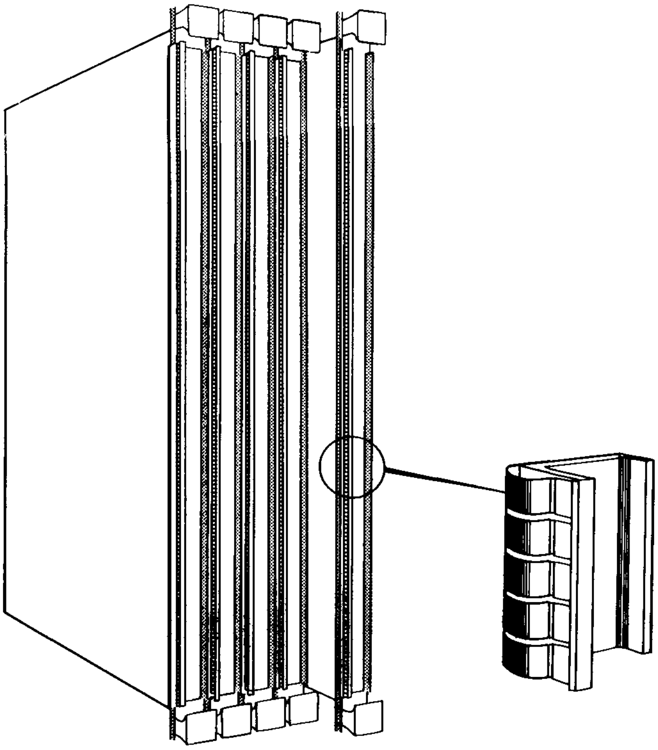


Fig. 4

2.2.2 Disc modules: The basic cabinet houses up to a maximum of nine 5·25″ peripheral devices, which may be a combination of fixed discs, optical discs, floppy discs and cartridge magnetic tape drives.

The exchangeable media devices are mounted at the top of the machine, directly behind the hinged flap on the front moulding, so that the operator can load/exchange media.

Fixed disc drives are mounted within bolt-in modules, which provide a flexible system of enhancement that avoids modification to the basic cabinet. Each module kit consists of a housing, disc drive, mounting brackets, power and signal cables, 'T' connectors and terminators to enable up to two drives to be fitted and connected into the basic system. The 'T' connectors enable the signal cables to be daisy-chained, and a terminator to be fitted.

Under consideration for the future are plug-in drive adaptors to eliminate the power and signal cables completely; this would reduce the time for installation and maintenance and increase reliability. See Fig. 5.

2.2.3 Cooling system: Basically, each heat-generating module is individually cooled (i.e. each module draws in air at ambient temperature). Cooling air is drawn in through the lower vents in the front and rear covers and is exhausted through the upper vents in the rear covers.

The fan module provides cooling for the 9U PCBs as well as the three

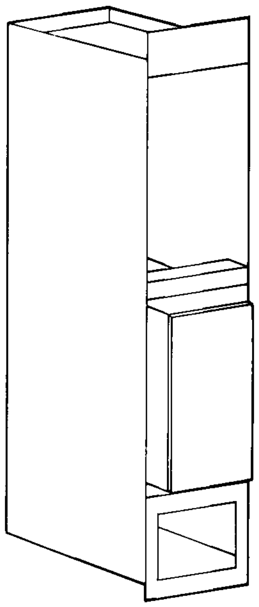


Fig. 5

peripheral devices mounted at the top of the cabinet. The 6U PCBs are cooled by a separate fan mounted at the front of the logic rack, which exhausts into a common plenum area within the fan module. Each disc module contains its own cooling fan.

Thermostats mounted above the logic rack monitor temperature within the machine and in the event of a system over-heat will instigate a controlled system shut down thus avoiding prolonged heat damage to sensitive devices.

2.2.4 Power supply unit: The power system for UNICORN was conceived as a single industry-standard package which derives all the necessary DC rails from the AC mains. Due to space constraints within the cabinet, switch mode techniques are necessary to achieve the required power conversion in the size, and weight, constraints imposed.

Sizing of the load produces a required power output of just under 1500 watts. The relatively high efficiency of switch mode power supplies ($>75\%$) means a manageable RMS current for normal domestic/office mains outlets within the UK. However, UNICORN is targeted at a worldwide market and a power system that accepts a range of input voltages and frequencies offers considerable manufacturing and servicing advantage. The big stumbling block in achieving this aim is the low single phase voltage (120 V) in North America, switchmode power supplies being tolerant to the frequency change between 50 and 60 Hertz.

A survey of electrical installations in North America reveals that phase-to-phase outlets are commonplace; a nominal voltage of 208 volts is available and applying tolerances to this and to the voltage ranges encountered in the UK, Mainland Europe, the Middle and Far East and Australia gives a working range of 180–264 volts, either 50 or 60 Hertz.

In order to give the UNICORN system a means of resilience to external AC power failures, an Uninterruptable Power Supply (UPS) is provided as an option. UPSs re-generate the AC voltage waveform from a static inverter. Usually, unless considerably over-rated for the job, UPSs are unable to supply large peak currents. But switchmode power supplies, by the very nature of their design, gave high repetitive peak currents, up to four times the RMS current, due to the repeated re-charging of the rectified mains reservoir capacitors every half cycle of AC input, and for the same reasons they have a high inrush current at switch on.

Both these characteristics are particularly unkind to a UPS. A market survey of UK power supply manufacturers showed that a number of the leaders were aware of this problem, and one manufacturer had a 1500 watt multi-output power supply in production with a front end “correcting” circuit which enabled a very near sinusoidal current to be drawn from the mains over the whole period of the AC cycle.

The overall strategy for UNICORN includes a facility for remote diagnostics. This facility is included in a central services module (CSM) within the system. This module is powered separately from the main system, and allows diagnostics to be run from a remote user port. As it is impossible to perform any sort of diagnostics on a system with a power supply partly or wholly shutdown because of a fault, a small (50 watt) power supply is provided to power a subset of the interface logic on the central services module and its associated cooling fan, and thus make it independent of the main system power supply.

2.2.5 Interconnect and Backplane:

Interconnect The overall philosophy adopted for UNICORN is to employ bus bars and/or woven pre-formed cableforms as much as possible. This enables whole assemblies to be manufactured off-line and simply installed in the cabinet, with the aid of Velcro, or similar, at final assembly.

The system signal loom is such an assembly and carries cabinet housekeeping, control information and auxiliary power. It consists of discrete wires woven together and pre-formed into the required shape, with all the ends having correct terminations fitted.

The five volt feeds to the backplane are similarly treated, and employ a moulded bus bar assembly which bolts directly onto the main power supply terminals. From this moulding, separate cables extend to the backplane in pairs, the pairs being separated by weaving techniques.

To interconnect the data interfaces of the several in-cabinet small computer system interface (SCSI) peripherals, a novel approach was adopted. A kit of three parts, namely a woven cable, a moulded 'T' junction, and a main SCSI terminator, are used to perform all interconnect functions. Each disc is fitted with a 'T' junction, which provides an additional two connectors at the rear of the drive. The first connector receives the incoming SCSI cable from the adaptor, while the second either daisy-chains to the next device in the chain or is fitted with a SCSI line terminator. For field upgrades, the terminator is removed from the current last disc and plugged into the 'T' junction on the new disc; a daisy-chain linker cable then connects the new disc into the system by plugging into the position previously occupied by the terminator. This arrangement allows very simple upgrading of an installation. See Fig. 6.

The DC power to the SCSI peripheral is routed from the power supply by bus bars formed of insulated copper laminates, the individual connections to the disc are effected by woven jumper cables. Where cables interfacing to external peripheral devices enter the cabinet a "window" is included in their construction to give all-round exposure to the overall braid screen of the cable. This screen "window" is placed under a clamping bar, which is secured to an integral part of the chassis, and thus effects a very positive method of strain relief while achieving the near 360 screen termination required to obtain a high level of resistance to static discharge and effective RF screening. This method is employed on all external interface cables.

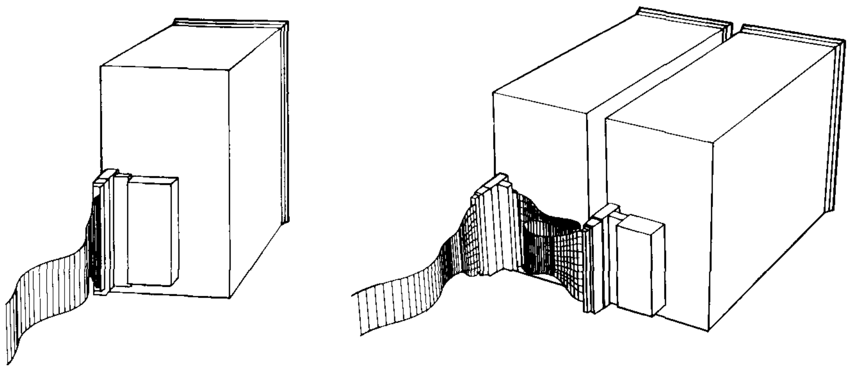


Fig. 6

Backplane The UNICORN backplane carries two bus systems:

- VMEbus
- A private bus of proprietary design (HSPbus)

It accepts up to a total of 20 PCBs. The first nine slots take triple-high (9U) Euro-card PCBs which interface to both bus systems. The remaining 11 slots take double-high (6U) PCBs conforming to the accepted VME standards, and thus only interface to the VMEbus.

Considerable thought has been given to the physical layout of the backplane, in two areas:

- The signal interface connector of the VME card slots
- DC power distribution.

For the former case, an interface connector (reverse DIN) is provided on the rear of the backplane to allow an interface card to plug up into the same slot as the VME card. It is envisaged that these cards be used for custom interfacing and providing extra facilities not normally obtained with a standard VME card. The adoption of this policy dictates the use of a backplane with "press fit" connectors. This is a new process to ICL, but one that has been widely used in the telecommunications industry for some years, so the technical risk is felt to be minimal.

Regarding DC power distribution the need to minimise the inductance of the 5 volt distribution is judged as vital, to ensure there is no "current starvation" when very high speed clocks are employed and/or large banks of memory chips are refreshed simultaneously.

To achieve this, four 2 oz copper power planes are employed, laid up alternately, to improve high frequency decoupling, and 10 000 μF of capacitance is distributed across the backplane. All the above requirements are achieved on a six layer, one piece backplane, with a characteristic impedance of the logic tracking complying with that defined in the VMEbus specification.

3 Standards and Approvals

UNICORN is aimed at a worldwide market, so compliance where many national and international standards are considered mandatory. Typical examples are:

- Underwriters Lab UL478
- Canadian Standards Assc. CSA C22.2 No. 220
- Federal Communication Commission (FCC) Regulation Part 15 subpart J class A
- Special Committee of the International Electrotechnical Commission, for Radio Interference (CISPR) Publication 22
- Verband Deutscher Eletrotechniker (VDE) 0871, 0805 and 0806
- International Electrotechnical Commission (IEC) 950
- British Standard 6301
- Local PTT Approvals

Many aspects of these standards are in-built into our own internal standards, and appropriate clauses regarding compliance with relevant standards always form mandatory clauses of purchase specifications of bought-out parts.

Complete machine approvals are carried out by extensive in-house testing and build appraisal, followed by a review of results by the relevant authority or submission to a recognised test house in the relevant country.

INTERFACES

The User System Interface—a challenge for application users and application developers?

Andrew T. F. Hutt
ICL HCI Strategy, Reading, Berkshire
Fiona Flower
ICL CPS Author Region, Kidsgrove, Stoke-on-Trent

Abstract

Over the last few years, the User System Interface has become an increasingly important component of any application. This paper describes the major trends in the development of User System Interface technology from the point of view of its impact on both the application user and the application developer. The paper concludes that the overall trends are that while users can expect their applications to become easier to use, applications developers have to face a major new set of challenges in order to design those applications.

Introduction

In its most basic sense, the User System Interface (USI) provides the means by which people and the computer communicate with each other. At a practical level, this interface is made up of a set of standardised elements and behaviour. At a theoretical level, it could be said to have three main components:

- The way the system communicates with the users
- The way users communicate with the system
- The users' expectations of the interface

The way in which the system communicates with users is controlled by the computer application. Thus, for example, an application controls how information is accessed, how computations are performed, and how the computed information is translated into a form which users can readily understand.

Users in turn have to be able to interpret the information presented to them by the system. They have to be able to understand what it means, and provide an appropriate response. The way in which they do this, using techniques such as key selection or mouse movement, represents the second aspect of the interface.

The third aspect may be summed up as the users' understanding of the whole process. Users have expectations of what a user system interface is, what it does, and how it works. Some of these expectations are formed by their experience with machines such as typewriters or calculators, as well as other computer systems. A good user system interface capitalises on this experience, and enables users to build on skills and knowledge they already possess.

The USI may therefore be described, in very broad terms, as the 'look and feel' aspects of an application: that is, what the application looks like to the user, and what it feels like to use. It should be stressed that this description does not formally extend to the ergonomic design of keyboard, mouse and screen, which are an integral part of the workstation engineering, and do not form part of the USI. In practical terms, the 'look and feel' aspects are represented by the visuals (the display items, such as forms, menus, context help or diagrams), and by the behaviour of these items (for example, whether the menu pulls down from an action bar, and what effect this has for the user).

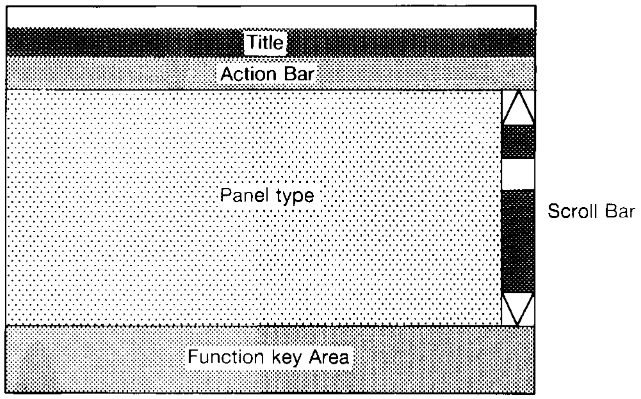


Fig. 1 Arrangement of display items on a screen

Figure 1 illustrates how display items might typically be presented to users through the USI.

The diagram shows groupings of information which might be displayed on the computer screen for a given application or service. Each of these groupings may be considered as a space which can be divided into smaller areas, each containing a particular type of information. The diagram identifies four main areas:

1 Application title
2 Action bar and pull-down menus
3 Panel body with scroll bar
4 Function key areas

Clearly, there are many factors influencing the presentation and behaviour of items in each of these areas: for example, the operating system under which the application or service runs, and other considerations, such as whether the input device is a keyboard or a mouse, or whether the application is character or graphics based. However, in general terms, these areas give a good indication of the type of items users are coming to expect from a computer interface.

The action bar, for example, is a useful device which provides visual cues to users to help them perform the desired action, without having to remember a name or type in a command. When users select an option, a pull-down menu appears, listing available actions. The user then selects the action he requires, using simple interaction techniques. An additional level can be provided by using pop-up windows which appear when users select an option from a pull-down menu.

Likewise, the function key area might be used to display a list of function key assignments. Users might be given the choice of either displaying this information in long or short form, or not displaying it at all.

The central area, or panel body, is tailored to meet the requirements of individual applications. It might have to be customised to suit specialised user actions, and may be used, for example, to display menus, forms, diagrams or charts. One of the key features of modern window management systems is that in general a panel body only contains one type of information.

Applications which need to display several types of information achieve this by using several different customised windows.

Why is the USI important?

In today's highly competitive industrial climate, customer organisations are looking more and more to Information Technology to give them strategic advantage over their competitors. This has led to a rapid growth of I.T. products, and consequently a greater choice for customers. As a result, organisations are in a position to demand systems which can be readily integrated into their business environment, and which will also prove acceptable to their users.

Indeed, user expectations have developed rapidly over the past few years, to a point where people are no longer prepared to spend time using products with poorly defined interfaces. Further, experience has proved that users find systems which support a common interface easier to learn and to use, since they are able to apply established techniques and skills to new applications. This has led to a marked preference for suppliers who can offer a consistent and coherent interface across a range of applications, products and services. As a result, recent years have brought an increasing trend towards specific USI styles (such as MOTIF, OPEN LOOK or MS/Windows) which offer just such a consistent approach. Organisations, in turn, have been able to reap the dual benefit of lower staff training costs, and more effective use of their I.T. products.

This growth in demand has resulted in an increased investment in user system interface technology. This is amply demonstrated by the fact that, in the last 4 years, major suppliers have all made major product announcements and are working to establish a set of open common standards for the user system interface.

Developments in User System Interface

Over the last few years, application users have seen User System Interface technology advance rapidly through four distinct stages, as illustrated in Fig. 2 below:

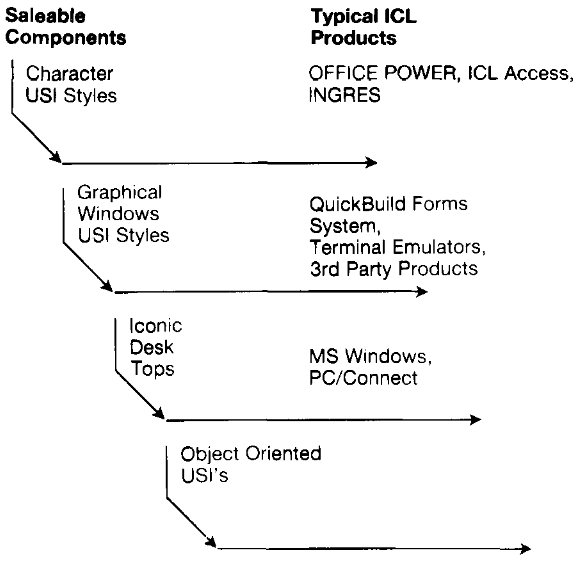

Saleable Components
Typical ICL Products
Character USI Styles
OFFICE POWER, ICL Access, INGRES
Graphical Windows USI Styles
QuickBuild Forms System, Terminal Emulators, 3rd Party Products
Iconic Desk Tops
MS Windows, PC/Connect
Object Oriented USI's

Fig. 2 Four stages in USI development

Character-based styles

The User System Interface has traditionally been interpreted through character-based styles of presentation. In past years major suppliers including ICL offered little consistency in style of presentation and many of the styles used were poorly designed or exhibited too many needless differences. Now, in common with the rest of the industry, ICL is working to reduce the variety of character-based styles to the minimum required to support its strategic products. As a result of this exercise ICL now has five character-based styles.

These styles have been vetted for consistency, and are defined in the ICL Register of USI Styles (see Ref. 4). All styles support USI objects such as forms, menus, context help and messages. Application development teams are being actively encouraged to develop products which support one or more of the registered styles. This should ensure that ICL's future strategic products support an interface which meets users' needs, is ergonomically sound, and at the same time exploits the capabilities of the workstations used to deliver it.

Graphical windows styles

The increasing popularity of IBM PCs and compatibles capable of supporting a graphical USI heralded the arrival of graphical windows styles. Some de facto standards have already been established in this area, notably Macintosh from Apple, OPEN LOOK from SUN, MOTIF from OSF, and Common User Access from IBM. ICL is investing heavily in applications which support graphical windows, and has decided to develop a range of software which will embrace one of the following industry standards for graphical styles.

- Windows
- Presentation Manager
- MOTIF
- OPEN LOOK
- Common User Access

Applications which conform to these styles provide users with USI objects such as windows, menus, forms and help and, in addition, graphical objects such as boxes, lines or circles. Each of the styles is accompanied by a style guide which defines how to design applications which conform to that particular style. It is worth noting that three of the graphical styles which have been adopted by ICL (Windows, Presentation Manager and MOTIF) are in fact broadly similar, resulting in what is effectively a consistent style across MSDOS, UNIX and OS/2 products supplied by a wide range of suppliers including ICL, IBM, DEC, Hewlett Packard and Microsoft.

Iconic desktops

Progress in the field of graphical USIs has led naturally to the third stage in USI development, which exploits the use of desktops. To date, there is a growing range of desktop products, all of which allow applications and files to be represented by icons. The major features supported by these products are all broadly similar. For example:

- When the user selects an icon, the application or file is activated, and control is transferred to the resulting application/file pair
- Icons can be grouped on the desktop to reflect the logical relationships between the objects they represent.
- Users are able to perform simple object manipulations (for example, throwing a document away by moving a document icon to the wastepaper basket icon)

The market already offers a range of desktop products. Choice between competing products depends on two things: the preferred USI style, and the level of integration between that particular desktop product and the local operating system and the supporting network management system.

Object oriented USIs

The next logical step to follow iconic desktops is to provide an object oriented user system interface. To date, perhaps the best example of an object oriented USI is Hewlett Packard's Object Management System (see Ref. 14). This system exploits both graphical windows and iconic desktops, but in addition allows application users to establish dynamic links between objects such as spreadsheets, charts and documents in such a way that changes to any one of these objects are automatically reflected in the others. For example, changing the numbers in the spreadsheet automatically leads to changes in the chart, or conversely, changing the shape of the chart causes the numbers in the spreadsheet to be altered.

USI: the application developer's view

From the above it is clear that customer pressure is forcing a trend towards graphical and object oriented USIs. Further than that, there is a marked tendency towards a growth of standards, and towards achieving compatibility and consistency across the range of styles available.

How do these trends affect application developers, in terms of the kind of systems they are asked to provide?

The first problem designers face is that, unlike traditional applications, graphical and object oriented applications interact with the user at a number of different levels.

At the visual level, the application provides a set of visual images which represent the objects in the application. To be effective, these must be readily identifiable by the user, and must trigger the appropriate mental and manual responses.

At a functional level, the application supports the user in carrying out a set of tasks. Within a graphical interface, support for these tasks is provided by the user through direct manipulation of visual images.

Finally, the application semantics need to be presented to the user in a way that reinforces effective use of the product. As an example, if a user mishandles a visual object, he should not be presented with a complex error message. Instead, the user action should be disallowed, and a low key error response such as a 'beep' should be used.

Figure 3 shows a suggested architecture for the development of applications with a graphical USI:

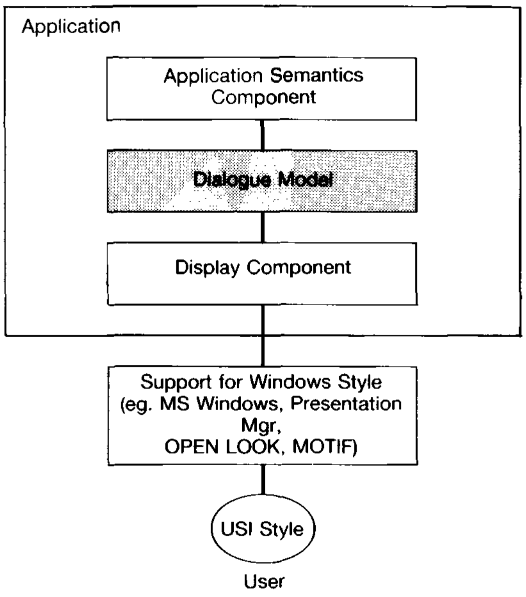

Application
Application Semantics Component
Dialogue Model
Display Component
Support for Windows Style (eg. MS Windows, Presentation Mgr, OPEN LOOK, MOTIF)
USI Style
User

Fig. 3 Architecture of an application with a graphical USI

The diagram illustrates the three parts a graphical application must contain if it is to support the three levels of interaction defined above. The three parts are:

- A dialogue model which represents the dialogue between the user and the system

- Display components, which present the application objects and the dialogue to the user via one of the standard USI enablers (MS Windows, Presentation Manager, OPEN LOOK or MOTIF)

- Application semantics, which represent the application rules

The development of an interface which meets these architectural requirements is one of the major challenges facing developers of graphical and object oriented applications.

One method of approaching this problem is to take a close look at the development cycle for graphical applications. Figure 4 below shows a representation of the process model for the application development cycle:

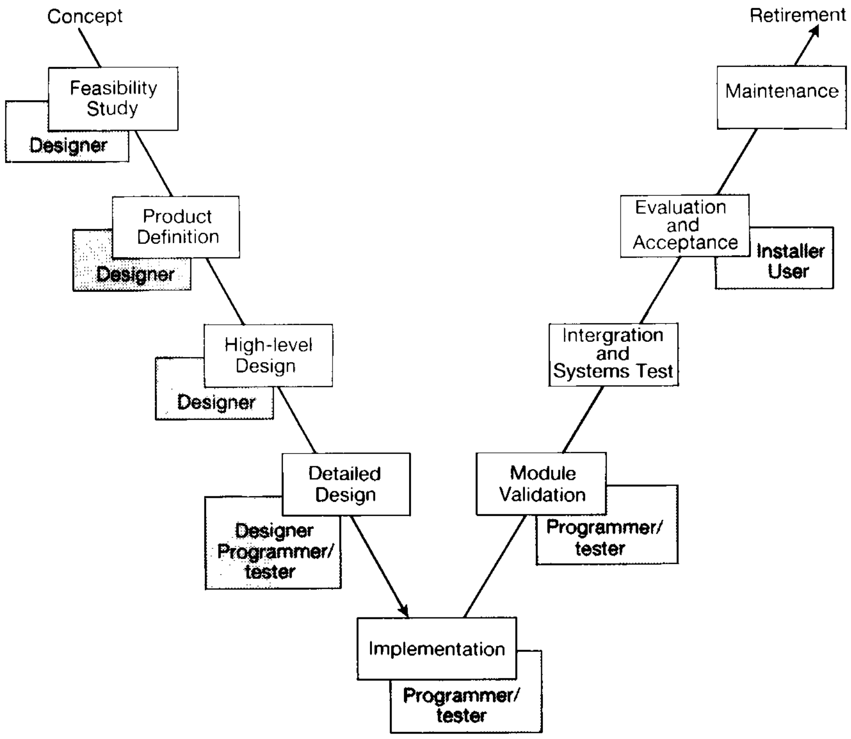


Fig. 4 Process model for the application development cycle

In the diagram, the clear boxes illustrate the steps in the development cycle; the shaded boxes show who is expected to perform the task involved at each stage in the development of an application.

One of the features of graphical applications is that they provide a much closer linkage between the user and the application. As a result, from the very beginning of the process (the feasibility study), designers need a clearer and more precise understanding of the user's tasks and working environment. The Marketing to Design workshops (see Refs. 1, 2 and 3) provide design teams with the skills they require to collect and handle this sort of information. The workshops aim to make sure that designers have a sound understanding of four essentials:

- the user of the application
- the job the user is doing
- the tasks the user has to carry out
- the objects used in these tasks

This information is then used to develop process, task and object models which represent the functionality to be supplied by the application. Designers then learn how to select the USI style which is best suited to deliver this functionality to target users.

At the High Level Design stage, the process, task and object models require to be turned into low level task descriptions which will act as a first foundation for the Detailed Design. Once this has been done, the designer or programmer should be in a position to develop an effective User System Interface for the product as illustrated in Fig. 5.

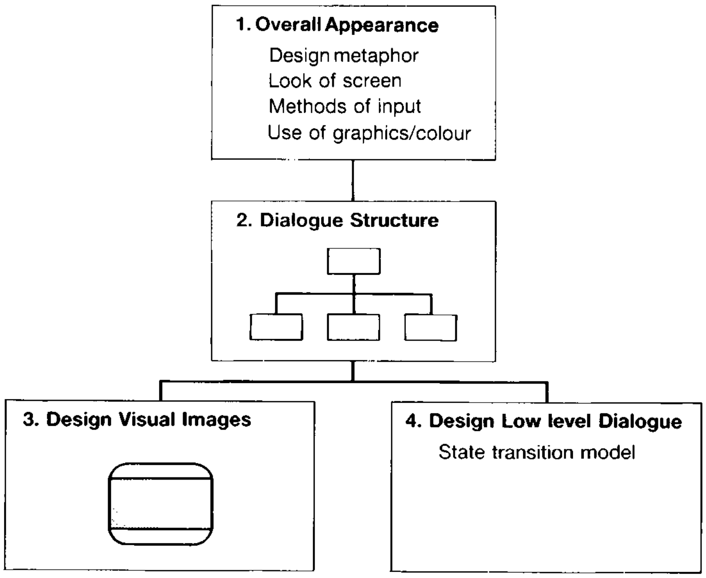


Fig. 5 Process model for developing a user system interface

There are four key stages in this process:

The first is to establish the overall appearance for the user system interface. This requires the designer to develop a design metaphor (such as the desktop of the Macintosh) for the application, to provide a conceptual basis for an interface to which users can readily relate, and, within that metaphor, to establish the broad parameters to which the interface must conform (e.g. the USI style, the input methods and the look of the screen).

Working from this basis, the next step is to define a dialogue structure based on the task model. This dialogue structure shows the overall flow of control and navigation at the user system interface. The lack of an explicit dialogue structure for an application can lead to the situation where users can easily become lost in a complex navigation structure which does not readily support the user's tasks.

Given the dialogue structure and the overall appearance of the interface, the next task is to define a set of visual images which represent the application objects, and which fit within the paradigm. The visual images are handled by the display component identified in the architectural model (Fig. 3), and are implemented using one of the standard tools which connect to the USI run-time systems (Windows, Presentation Manager, MOTIF or OPEN LOOK).

The final task is to define a low level dialogue model, which implements the dialogue structure. This model defines, for example, how the user will manipulate the visual images. At present, dialogue models are implemented using programs written in C or C^{++}. It is expected however, that these models will be implemented in Dialogue Description Languages such as SET and Actor. If the application is to be an object oriented application which is integrable with the Hewlett Packard Object Management System, the dialogue model should conform to the rules of the HP Dialogue Description Language (see Ref. 5).

Conclusion

From an application user's point of view, the User System Interface has progressed rapidly over the past few years, as traditional character-based styles are being replaced by graphical and object oriented USIs. These advances have been welcomed enthusiastically by users, who find the new styles generally more attractive, more consistent, and easier to learn and use. As a result, the demand for products which exploit graphical USIs is growing day by day. In order to meet this demand, application developers now have to face a major new set of challenges, not least of which is the need to come to terms with and exploit a whole new concept in architectural design.

References

1 HUTT, A.T.F.: USTM Final Report. 1988 available from ICL HCI Strategy.
2 HUTT, A.T.F.: Describing a Product Opportunity. Brochure, available ICL Internal Staff Training, 1989.
3 HUTT, A.T.F.: How to Identify a High Valued Solution. Brochure, available ICL Internal Staff Training, 1989.

4 HUTT, A.T.F.: The ICL Register of USI Styles. ICL HCI Strategy Internal Report, HCI/SAM/259, 1989.
5 HEWLETT PACKARD: HP New Wave Environment—Programmer Reference Manual. Manual Part No. D1710-90001, Hewlett Packard, Santa Clara, 1988.
6 IBM: Common User Access - Panel Design and User Interaction. IBM technical publication SC26-4351-0, 1988.
7 MICROSOFT CORPORATION: Microsoft Windows - Software Development Kit - Application Style Guide, 1988.
8 SUN MICROSYSTEMS INC.: OPEN LOOK Graphical User Interface Functional Specification. Technical Publication 800-4085-01, 1989.
9 X-OPEN COMPANY LIMITED: X/Window System: X/Open Portability Guide, Volume 6, 1988 Edition. (Prentice Hall, Inc., USA, 1988).
10 P.A. CONSULTANTS LIMITED: SET Tools Technical Overview, 1989.
11 DUFF, Charles *et al.*: Actor Language Manual (The Whitewater Group, Inc. Evanston, 1987).

The emergence of the separable user interface

Ernest Edmonds
LUTCHI Research Centre, Loughborough University of Technology, UK

Abstract

The meaning of the term "user interface" is briefly reviewed and the concept of this as an identifiable component in a system is explored. The history of the development of separable user interfaces is reviewed, together with the growth of interest in user interface software systems and in techniques for user interface specification. Separable user interface architectures and practical problems that have arisen in the use of separable user interfaces are discussed.

Introduction

The user interface to a computer system goes under a number of names including the human-computer interface, the computer-human interface, the front end and the man machine interface. Whilst "front-end" tends to be used in the restricted domain of so called "intelligent interfaces" and "man-machine interface" is sometimes used to refer to systems that do not involve computers, for the purpose of this paper they can all be considered to be synonymous. Many papers and books on the user interface exist, including ones that give detailed advice on how to design and how to construct them. Nevertheless, there is still no universally agreed definition of the user interface. For example,

> "The human interface with computers is the physical, sensory and intellectual space that lies between computers and ourselves". (Bolt, 1984),
> "The user interface is that part of a system that the user comes in contact with physically, perceptually, or conceptually." (Moran, 1980) or
> "The functionality defines what the program can do, and the user interface defines how users tell the program what to do, and how the program tells users what it did." (Szekely, 1988).
>
> In fact many authors omit a definition altogether (e.g. Shneiderman, 1987).

In English, the normal use of the word "interface" is to name the relationship between two entities, particularly where that relationship is well defined. Increasingly, however, the term "interface" has been applied to special devices that enable one of the objects to hold the given relationship with the other. Thus, for example, one might have to buy a special "interface" in order to enable one's computer to operate a particular printer. A similar tendency has occurred in the case of the user interface. The study of the relationship between the user and the computer is, now, most frequently known as the study of human-computer *interaction.* The user *interface* normally refers to the hardware and software that enables the computer to play its part in the interaction. It is widely assumed that it is possible to identify this interface as part of the system and, conversely, that it is possible to identify components that perform functions that should not be seen as part of the interface. It is this assumption that is the central concern of this paper.

Early work

Without doubt, the most important early work in this field was done as William Newman. This work built upon Sutherland (1965) in important respects but, nevertheless, represents the first significant steps towards the current concern for the object known as the user interface.

Newman (1968a) describes a virtual input device that enabled the user to enter numerical data on a continuous scale with the aid of a light pen or a similar device. Newman terms his logical device a "light handle" because it can be seen as a simulation of a handle that can be wound in order to increase or decrease the associated numerical value. The significance of this work was that it demonstrated the possibility of constructing what, today, might be known as an "interaction object": that is to say, a self-contained software module that uses the physical input/output devices available in order to enable the user to provide some particular kind of input to the machine. Whilst, in modern terms, this might be seen to be a very low level of separation, it represented an important conceptual step.

The most significant early work was, however, Newman's system for interactive graphical programming (Newman 1968b). A *network definition language* was introduced that partly defined interactive software in the form of state diagrams. A compiler for this language could generate run time code. This system was based upon a distinction between the *reaction handler*, with its associated tables, and the *procedure component*. The network definition language was compiled into the tables that the reaction handler used, as it were, to run the user interface. The procedure component was written in a conventional language that did not deal directly with any interaction with the user. The mechanism for linking the two aspects was the insertion of program blocks into the definition of each appropriate state of the state diagram. Thus, the network definition language defined the user interface together with its links to the functional part of the system: the procedure component.

Whilst Newman's Light Handle and his Reaction Handler both demonstrated the separation of user interface issues from functional ones, the distinction between them, in this respect, is very important. The separation offered by the Light Handle is that of an interaction object that can be evoked by the functional code but that looks after itself once active. The Reaction Handler is quite different in that it is in control of the system and evokes functional actions as required. It is, therefore, an example of a separate user interface. The paper did not, however, address the issue of the precise scope of concern of the procedure component. Indeed, at that time, this was perhaps not a major issue but more recently, as will be discussed below, this has become a significant point of debate.

Parnas (1969) also proposed the use of transition state diagrams for the representation of dialogues, although he only advocated them for the "top level" of user interface design and so, in his work, they did not lead directly to implementation. The links between the user interface and the functional components of the system were not associated with states, as in Newman's case, but with the arcs. The method was to label each arc with an input/output pair, representing the user input that would cause the arc to be traversed and the output that would result. Only simple examples are discussed and, as with Newman, no detailed discussion of the issues of user interface/functional component separation is given.

Denert (1977), in considering the proposition described above, introduced two more innovations. Firstly, he recognised the computational limitations of state diagrams, which represent finite state machines, and proposed the use of recursive state diagrams, which are much more powerful. Secondly, he dealt with the interconnection of the user interface and the functional component by introducing "task nodes" into the diagrams, which call the functional code. In some respects, this is similar to Parnas' scheme but it adds the possibility of different branching depending on the outcome of the functional operation. Thus we now have the concept of the functional code changing the course of the dialogue. It would seem that for the separation of the user interface from the functional code to be a reality for complex systems, both of Denert's innovations are required.

First generation user interface systems

During the 1970's a number of research workers followed William Newman's early example and constructed user interface software systems that, in various ways, provided for a separation between the user interface and the functional code. Stalling (1974) demonstrated a system for generating dialogue code that worked somewhat after the manner of a report generator program. Florentin (1977) elaborated upon this and noted the importance of the linking of the dialogue code to the rest of the system. However, neither author added significantly to the specific discussion of separability.

Maher and Bell (1977) specifically proposed a virtual machine as the key user interface module. They illustrated their work with a user interface to a database and briefly discussed some problems that arose with the separation of concerns. In particular, they found that it was convenient to organise the database in a particular way for the benefit of the user interface, but that this conflicted with other requirements.

Black (1977) described another system which provided a separate "dialogue processor" which managed the user interface. A dialogue specification language was provided that compiled into a form directly usable by the dialogue processor. In this system, the dialogue processor produced a "target file" which contained the input required by the functional component. Clearly, this was a simple view by today's standards and so, again, problematic issues of separation were not addressed.

State diagrams are popular for the specification of user interfaces, but several other formalisms are also used. Green (1986) suggests that one of the earliest examples of a grammar-based user interface system was that reported by Edmonds and Guest (1977). It showed how a grammar-based language could be used to define an interface flexibly. It was used to construct a programming tutor and so was demonstrated as a feasible system. However, it is again true that the simplicity of the separation between user interface and functional system was such that the issues often seen now as the problems of separability were not discussed.

Ian Newman (1978) demonstrated how separation could be used to provide a variety of interfaces to a single system. Practical work, again, was done to prove the point but, as before, the particularly difficult issues of separability that are perceived today were not considered. Lafuente and Gries (1978) extended Pascal in order to provide what they called "dialogue frames", each of which could include a procedural block. Thus the structure of this user interface was somewhat similar to Newman's (1968b). The difference was that Lafuente and Gries provided an extension of a standard programming language, whereas Newman kept his interface code quite separate from the functional code.

User interface specification

Whilst the work on specifying user interfaces has not in general addressed the issue of separability directly, it is quite clear that the whole enterprise is founded upon the notion of separability. If it were not so, what would it be that one was specifying?

An important model for these specifications was Moran (1980) in which he proposed a view of the user interface that had a number of levels incorporated into three components: physical, communication and conceptual. A key point is that, in this view, only the conceptual component needs direct contact with the functional aspects of the system. In Moran's view the conceptual component contained two levels. The *task level* is concerned with the structuring of the task domain, for the benefit of the user, and the *semantic level* is "built around a set of objects and manipulations of those objects. To the system these are data structures and procedures; to the user they are conceptual entities and conceptual operations on those entities." (Moran, 1980). The semantic level was seen as a way of representing the system's functional capability. This particular aspect of the structure of a user interface has proved quite fruitful in attempts to clarify the separation from the functional code. This paper, and its elaboration (Moran, 1981), proposed that the user interface could be specified in a well-defined set of levels. This work has been influential both in the study of user interface specification and, more specifically, in the study of architectures for separable user interfaces.

Mark Green (1981) reported a notation for the specification of user interfaces and presented it in the context of a traditional software development model. Whilst that model is no longer necessarily accepted (see for example Hekmatpour and Ince (1987) or an earlier questioning by Edmonds (1974)), the specification ideas are not invalidated. Green's notation includes the notion of atomic objects and operations that correspond to Moran's objects and manipulations. These represent the lowest level of the system of concern to the interface. Beyond that, is the concern of the functional code. This has been the standard view of separation. In his paper, however, Green introduces an important additional property at this level. He allows for the specification of *invariants*. These are relationships which must hold between the operators and objects. Whilst he only gives a simple example of an invariant, it is clear that it is entirely unrealistic to consider these objects as independent and that quite complex invariants may be required. The only question at issue is whether the invariants can and should be seen as the concern of the user interface.

Edmonds (1981) consolidated earlier work and put forward a specification notation that combined state diagrams with grammars, each having their appropriate role. This work continued the tradition begun by William Newman in that a compiler was available for the notation. The links between the user interface and the functional code were provided partly in the manner of Parnas, as described above, except that the input/output specifications on an arc could have a complex transformation relating them, so that specific inputs in the appropriate class could generate relevant outputs. The outputs were passed to the functional code at a "task node", as in Denert.

Naturally, the early work in the field did not address the issue of specifying user interfaces that handled direct manipulation specifically. However, Jacob (1986) has shown how the methods that employ state diagrams can accommodate direct manipulation.

Architectures

Following Moran's work, discussed above, much debate has taken place about appropriate architectures for separable user interfaces and a large

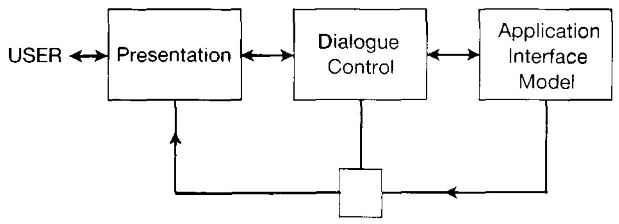


Fig. 1

proportion of the issues of concern have been in the support for separability. In many ways the details of the architecture can be seen as a precise definition of the scope of the user interface.

A very important event in the development of separable user interface architectures was the workshop held in 1983 in Seeheim (Pfaff, 1985). One group in the meeting, consisting of Jan Darksen, Ernest Edmonds, Mark Green, Dan Olsen and Robert Spence, developed an architecture for the user interface (Green, 1985), see Fig. 1.

As Löwgren (1988) points out, however, the first proposal of a user interface software architecture of this form was probably that of Edmonds (1982), see Fig. 2. It built directly upon Moran's concept of layers. In this view the separation between user interface and functional code was precisely at Moran's semantics level where the objects and manipulations that can be seen as atomic, from a user's point of view, reside.

The Seeheim application interface model went further than either Edmonds' or Green's earlier proposals. The idea was that it might contain, for example, information about the effects of operations in order to assist the user interface in the provision of help. Also, in order to deal with larger amounts of data generated by the functional code, such as dynamic images, a direct

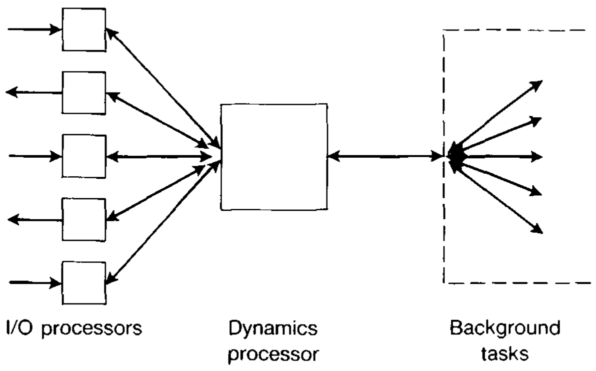


Fig. 2

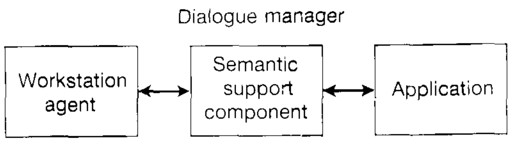


Fig. 3

path to the display was proposed. In this case, the only functions of the user interface were to open such a path and to decide where it should point to on the screen. This architecture has become widely known as *the Seeheim model* and a good deal of the subsequent debate has used it as the base reference. It is, perhaps, not surprising that this model has been so important because, apart from the earlier architecture from Edmonds, position papers were presented at Seeheim by Thomas and Olsen, both of which proposed architectures in the same spirit as what became the Seeheim model.

Many papers have been written concerning modification to this model. For example, Dance et al (1987), see Fig. 3, and Edmonds and McDaid (1990), see Fig. 4. It is noticeable, however, that most are variations on the Seeheim view, rather than radical departures from it. We may assume, therefore, that the climate of opinion remains to favour something of this flavour as an architecture for the separable user interface. In particular, it can be seen that the user interface is seen to include a substantial part of the software of a system.

Practical issues of separability

Certain position papers presented at Seeheim were already addressing practical separability issues. In particular, Olsen (1985) and Sibert et al (1985) consider interdependencies between the layers with some care. One of the working groups, in fact, considered the issue specifically, as reported by

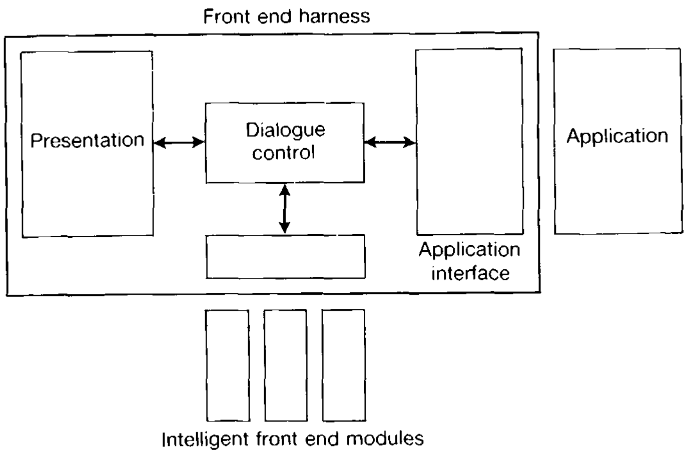


Fig. 4

Enderle (1985). Their basic conclusion was that separability, even for CAD/CAM applications, appeared to be possible given careful design. The group looked particularly at the services provided by the user interface to the functional code and vice versa. This was probably the first attempt to catalogue these difficult concerns (see Table 1).

Szelsky, in his PhD thesis (1988), considered separability in detail. He considered the interface as, perhaps, a slightly shallower entity than others have done, but this enabled him to identify clearly a set of interaction types. This classification is itself a contribution to the definition of the scope of the user interface. Even if one does not wish to limit ones view of the interface in Szelsky's way, it is clear that his work is valuable as a contribution to our understanding of that part of the user interface termed the "presentation layer" at Seeheim.

Alty and McKell (1986) have made a valuable study of the application model in the Seeheim sense. They demonstrate an application model for a command driven system and show how a study of interaction sessions can help to identify the knowledge required in the model. A particular outcome of the empirical study was that they identified a need for a planning component in the application model. Thus, whilst they confirmed the viability of the separable user interface, they demonstrated in practice that the interface can require quite sophisticated components within it.

In another practical case study, Manheimer et al (1989) showed that the separable user interface is a quite realistic concept. It is of interest to note that the mechanisms for coupling user interfaces and functional components vary widely and that, in this case, more than one mechanism was found to be necessary. In applying their LUIS interface system they found the need for both "tightly" and "loosely" coupled relationships. "The tightly coupled routines are linked with the LUIS executable image. A subroutine interface is available for these routines to communicate with the LUIS interface manager, i.e., to send messages to manipulate the ongoing dialogue and to receive messages that inform the application of the state of the dialogue. When a tightly coupled routine is invoked, the user interface is suspended until the application has completed its processing." (Manheimer et al, 1989).

In another practical investigation, Dance et al (1987) encountered the difficulty of rapid semantic feedback in a separable user interface much as discussed at Seeheim (Strubbe, 1985). If one considers a screen displaying three things, a window, an icon for a file and an icon for a disc, then consider the difference between the following two situations. If the user drags the window over the disc icon, the icon is simply obscured. If the user drags the file icon over the disc icon, then the latter is, typically, highlighted. The difference represents more or less instant semantic feedback.

It is sometimes argued that this semantic feedback problem is a case against separability, but in fact that case is far from clear. It is true that it would be

Table 1 Services at the application – UIMS interface

Services of the UIMS for the application	*Services of the application for the UIMS*
* Give a value matching constraints	* Give a value (e.g., for use in help functions)
* Switch input sources (e.g. between user & journal)	* Give a value to be used instead of user input
* Tell the user a value	* Perform graphics update
* Display some graphics	* Set application state variable to a given value
* Handle an application error	* Grant permission to change application data
* Set ouput context	* Grant permission to change application data
* Set dialogue scope	* Give pick support
	* Give task status
	* (Re-) Set task status to a (previously saved) state
* Undo something	* Undo something

hard to achieve an acceptable performance if tokens had to be passed through the layers, to and from the application, in order to achieve it. However, there is no reason why the presentation layer itself should not contain the appropriate knowledge. After all, the example given above does not imply any application action.

This difficulty may have arisen because many researchers have been influenced by developments in computer graphics where closed packages are often used for implementing interactions. As even the simple example above makes clear, all levels of a separable user interface must be programmable in the sense that the system developer may need to build application specific knowledge into any part of it. This is not an argument against separability, of-course, although it suggests caution about the utility of completely general purpose user interface modules.

The difficulty here is whether the user interface might need too much knowledge (Strubbe, 1985). However, nobody has defined "too much" in this context. Even as early as 1978 Sutton and Sprague found that more than half of the code in interactive business applications was devoted to the user interface. Size is not the problem, although duplicating application code within the interface would be, of course. It would seem that where functional code has to be entered in order to generate the semantic feedback and where that feedback is required very quickly indeed, or where it involves large volumes of data, either the direct path, as proposed at Seeheim, or a closely coupled link, as in Manheimer et al, could be used. However, semantic feedback remains an area for investigation.

Conclusions

Issues of separability still remain to be resolved but there is enough evidence now that the separable user interface can be defined. Perhaps the biggest problem that has stood in the way of successful work in the area is that people have sometimes attempted to build general purpose front ends in the belief that (using the Seeheim terminology) only the application model needs to contain knowledge of application issues. In point of fact, even the presentation layer must often be programmed with specific application dependent knowledge. The separable user interface is not at all ignorant of the functions of the system.

If we return to the definitions of the user interface quoted at the beginning of this paper, it is interesting to notice that the separable user interface can be seen as a concrete elaboration of each of them. Moran (1980) went on to propose that "Any aspect of the system that shows through to the user and enters the user's model is a part of the user interface". The emergence of the separable user interface has occurred partly as an attempt to represent everything "that shows through to the user" in an identified software module. The degree to which this attempt has been successful remains to be fully assessed, but it is quite clear that the study of separable user interfaces is becoming increasingly important in concentrating our attention on some of the most significant aspects of interactive systems.

Acknowledgement

The preparation of this paper was partly funded by the Alvey programme, grant reference MMI/062, and the ESPRIT2 FOCUS project, reference 2620.

References

ALTY, J.L. & McKELL, P. (1986). Application modelling in a user interface management system. In People & Computers: Designing for Usability. Eds. M.D. Harrison & A.F. Monk. pp. 319–35.

BLACK, J.L. (1977). A general purpose dialogue processor. Proc. Nat. Computer Conf. pp. 397–408.

BOLT, R.A. (1984). The human interface. Lifetime Learning Publications.

DANCE, J.R., GRANOR, T.E., HILL, R.D., HUDSON, S.E., MEADS, J., MYERS, B.A. and SCHULERT, A. (1987). *The run-time structure of UIMS-supported applications.* Computer Graphics 21, 2, pp. 97–101.

DENERT, E. (1977). Specification and design of dialogue systems with state diagrams. Proc. Int. Computing Symposium, Liège, pp. 417–23.

EDMONDS, E.A. (1974). A process for the development of software for non-technical users as an adaptive system. General Systems, 19, pp. 215–17.

EDMONDS, E.A. & GUEST, S.P. (1977). An interactive tutorial system for teaching programming. IERE Proc. 36 – Computer Systems and Technology, pp. 263–70.

EDMONDS, E.A. (1981). Adaptive man-computer interfaces. In Coombs & Alty (eds.), Computing Skills and the User Interface. Academic Press, pp. 389–426.

EDMONDS, E.A. (1982). The man-computer interface: a note on concepts of design. Int. J. Man-Machine Studies, 16, pp. 231–36.

EDMONDS, E.A. & McDAID, E. (1990). An architecture for knowledge-based front-ends. Knowledge-Based Systems. (To appear).

ENDERLE, G. (1985). *Report on the interface of the UIMS to the application.* In G.E. Pfaff (ed), User Interface Management Systems. Springer Verlag, pp. 21–9.

FLORENTIN, J.J. (1977). Automatic generation of interactor programs. Proc. Int. Conference on Displays for Man-machine Systems, Lancaster. IEE pub. no. 150, pp. 126–9.

GREEN, M. (1981). A methodology for the specification of graphical user interface. Computer Graphics, 15, 3, pp. 99–109.

GREEN, M. (1985). Report on dialogue specification tools. In G.E. Pfaff (Ed), User Interface Management Systems, Spring Verlag, pp. 9–20.

HEKMANPOUR, S. & INCE, D.C. (1987). Evolutionary prototyping and the human-computer interface. In: H.J. Bullinger & B. Shackel, eds. Proc. of INTERACT 87. Amsterdam: North Holland; IFIP, pp. 479–484.

JACOB, R.J.K. (1986). A specification language for direct manipulation user interfaces. ACM Transaction on Graphics. 5, 4, pp. 283–317.

LAFUENTE, J.M. & GRIES, D. (1978). Language facilities for programming user-computer dialogues. *IBM J Res. Develop.* 22, 2, pp. 145–58.

LÖWGREN, J. (1988). History, state and future of user interface management systems. SIGCHI Bulletin, 20, 1, pp. 32–44.

MAHER, P.K.C. & BELL, H.V. (1977). The man machine interface – a new approach. Proc. Displays for Man-Machine Systems. IEE Publication 150, pp. 122–5.

MANHEIMER, J.M., BURNETT, R.C. & WALLERS, J.A. (1989). A case study of user interface management system development and application. Proc. CHI'89, pp. 127–32.

MORAN, T. (1980). A framework for studying human-computer interaction. In Guedj, R.A., Ten Hagen, P.J.W., Hopgood, F.R.A., Tucker, H.A. & Duce, D.A., Eds., Methodology of Interaction, Amsterdam: North-Holland, pp. 293–302.

MORAN, T. (1981). The command language grammar, a representation for the user interface of interactive computer systems. Int. J. Man-Machine Studies, 15, pp. 3–50.

NEWMAN, W. (1968a). A graphical technique for numerical input. Computer Journal, 11, pp. 63–4.

NEWMAN, W. (1968b). A system for interactive graphical programming. Proc. of Spring Joint Computer Conference, pp. 47–54.

NEWMAN, I.A. (1978). Personalised user interfaces to computer systems. Proc. of the European Computing Congress, London, pp. 473–86.

OLSEN, D.R. (1985). Presentational, syntactic and semantic components of interactive dialogue specifications. In G.E. Pfaff (Ed.), User Interface Management Systems. Springer Verlag, pp. 125–33.

PARNAS, D.L. (1969). On the use of transition diagrams in the design of a computer interface for an interactive computer system. Proc. Nat. ACM Conference, pp. 379–85.

PFAFF, G.E. (Ed.) (1985). User interface management systems. Proc. of the Workshop on User Interface Management Systems, Seeheim, 1983. Springer Verlag.

SHNEIDERMAN, B. (1987). Designing the user interface. Addison-Wesley.

SIBERT, J., BELLIARDI, R. and KAMRAN, A. (1985). Some thoughts on the Interface Between User Interface Management Systems and Application Software. In. G.E. Pfaff (Ed.), User Interface Management Systems. Springer Verlag, pp. 183–92.

STRUBBE, H.J. (1985). Report on role, model, structure and construction of a UIMS. In G.E. Pfaff (Ed.), User Interface Management Systems. Springer Verlag, pp. 3–8.

SUTHERLAND, I.E. (1965). SKETCHPAD: A man-machine graphical communication system. MIT Lincoln Lab, Lexington, Mass. Tech. Report TR-296.

STALLING, W. (1974). Some aspects of an interactive dialogue-generation facility. Proc. Intl. Conf. on Systems, Man and Cybernetics, pp. 495–498.

SUTTON, J. and SPRAGUE, R. (1978). A study of display generation and management in interactive business applications. Tech. Report RJ2392 (31804). IBM San José Research Laboratory.

SZEKELY, P. (1988). Separating the user interface from the functionality of application programs. CMU-CS-88-101. Carnegie-Mellon University.

SMIS – A Knowledge-Based Interface to Marketing Data

Chris Dobbyn
Technical Consultant, ICL Retail, Bracknell, Berkshire
John Cheesman
Senior Research Engineer, STC Technology Ltd, Harlow, Essex

Abstract

The SMIS (Strategic Marketing Information System) project arose from an ESPRIT project (P1098), the aim of which was to develop a methodology for the development of knowledge-based systems (KADS). This paper discusses the KADS background to SMIS, and then outlines the evolution of the SMIS project itself. SMIS is a system the function of which is to allow a marketeer to extract information from a range of data sources such as files, databases, etc. without any knowledge of query languages or the underlying structure of the data s/he is addressing. Queries and the information generated from them by SMIS are expressed in terms that are meaningful to the user. The high-level design of SMIS is discussed, and the paper concludes with an evaluation of SMIS and some reflections on the future of the system and of systems like it.

1 Introduction

The origins of SMIS (Strategic Marketing Information System) lie in the ESPRIT project P1098 – Knowledge-Based Systems Methodology, a collaboration between STC Technology, SD-SCICON, SCS GmbH, KBSC, CAP Sogeti Innovation, and the University of Amsterdam. The aim of P 1098 was the development, from a theoretical basis, of a methodology for the development of commercial knowledge-based systems. This methodology, known as KADS (Knowledge Acquisition and Design, Structured) aimed to provide a number of tools, processes and techniques from which the user could select those most appropriate to his/her own domain. By the time of publication, this project will have been completed.

Associated with the main body of KADS work were a number of experiments and case-studies, whose aim was to test the viability of KADS as a system development technique by applying it to real problem domains. Of particular interest to the KADS workers at STC Technology were *generic systems*, systems which stood between the (often massively expensive), "one-off" bespoke expert systems of the early days of knowledge-based technology, and general purpose tools such as expert systems shells, which are reusable, but less powerful. Since experience has proved it futile to look for a significant set of common approaches between practitioners in most commercially interesting domains, *general purpose* systems in such areas are infeasible. However, a *generic* system was seen rather as an architecture, a set of core functionalities and a reusable, tailorable software library, from which systems answering specific user demands could be built. SMIS (Strategic Marketing Information System) started life as KADS Experiment F8, a generic system to assist market analysts.

The chosen domain of SMIS was Retail, a major ICL vertical; the aim of the experiment was to design a system that would provide intelligent assistance to marketeers. Strategic marketing is a complex process involving the gathering, cross-referencing and analysis of large quantities of data, and developing from this data an understanding of customers, competitors, trends, etc. Companies accumulate vast amounts of data in corporate databases, files, EPOS systems, etc. and much more is available in external databases, etc. Thus the strategic marketing activity may be seen as breaking down into two main activities: the location, collection, integration and cross-referencing of appropriate data; and the carrying out of a number of analysis tasks on that data. The aim of SMIS is to automate the first process and to provide assistance in the second. Clearly, any system that can provide such support must carry a range of "knowledge": of marketing activities, of marketing concepts, of information sources, of statistical analysis techniques. Hence, it is appropriate to think of SMIS as a knowledge-based system; further – in keeping with the theme of this edition of the Technical Journal – it is also apropriate to think of SMIS as an *interface* between marketeer users and a range of data sources.

The architecture and internal processes of SMIS will be discussed in some detail in Section 3, below; and an evaluation of the current state of the project in Section 4. Before that, however, we present a brief outline of KADS.

2 KADS

Until quite recently, much of the development work in KBS has been carried out in universities and other research and academic institutions; and the prevailing development paradigm has been that of prototyping and incremental development paralleling a knowledge acquisition process. KADS, with its focus on the development of commercially viable knowledge-based systems, breaks with this tradition. Four key elements in the KADS methodology are:

1 External requirements capture
2 Knowledge acquisition techniques
3 Separation of analysis, design and implementation
4 A conceptual model

Each of these features is now briefly discussed.

2.1 KADS – external requirements capture

The systematic analysis of client needs and organisational background is a standard feature of the development of conventional software systems; knowledge-based system developers, however, have tended to ignore, or place little weight on, this activity. Clearly, in any commercial context, systems must operate within an operational environment with real users, must answer a proven need and must be cost-justified if they are to be developed at all.

In the KADS methodology, therefore, requirements analysis is an integral part of the development of the conceptual model, which is a central feature of the KBS development process as a whole. A distinct stream in the analysis phase is devoted to an "external view" of the system, and a number of techniques, focussing on organisational objectives, functional problems, system objectives and compatibility requirements, have been developed to facilitate this analysis stream. A detailed description of these techniques would take this paper beyond its intended scope: the reader is referred to (Barthelemy et al 1987).

2.2 KADS – knowledge acquisition techniques

In KADS, the knowledge acquisition process is based on established techniques of interviewing an expert. However, the process is *model-driven* rather than data-driven; that is, a set of initial interviews is used to construct a provisional model of the problem solving process, selected from a library of generic models, which is then used to focus and direct subsequent interviews, in the course of which the model is refined and/or adapted.

The output of the activities described in 2.1 and 2.2 will be, among other documents, a *requirements specification* and a *conceptual model* of the problem solving process.

2.3 KADS – separation of analysis, design and implementation

Traditionally, knowledge-based systems have been developed by means of a process in which analysis and implementation run side by side. KADS proposes a life-cycle model in which the analysis, design and implementation phases are separated from one another, as is the case with conventional software development methods. Such an approach allows much greater flexibility: it is easier to estimate feasibility and cost benefits; it is easier to control budgets and to plan; developers have greater control over the implementation process and, ultimately, system maintenance is facilitated.

2.4 KADS – the four-layer conceptual model

The conceptual model is a description of the subset of expertise that is to be embodied in the proposed system. This model is divided into four related layers, as in Fig. 1.

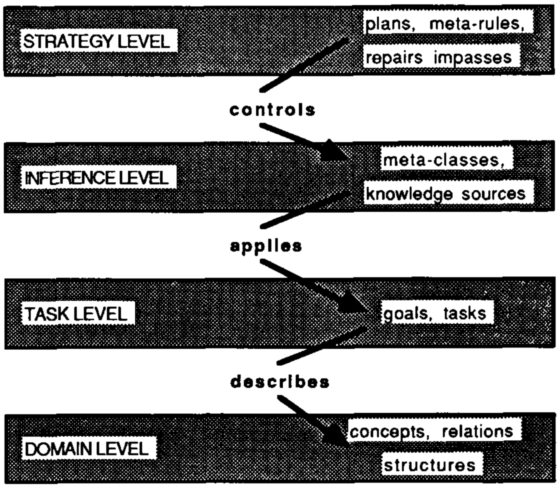


Fig. 1 KADS four-layer model

The lowest level of description is the domain layer, in which knowledge of the basic objects of the domain is represented. The inference layer incorporates knowledge about what inferences may be made given the knowledge expressed in the domain layer. The inferencing procedures are referred to as *knowledge sources* and the concepts on which they act, which are domain level entities or combinations of them, are named *meta-classes*. These are combined in the task layer, where knowledge of how inferencing is carried out in the problem solving context is expressed in the form of goals and tasks. The strategic layer represents the knowledge required to plan, guide and monitor the problem solving process.

2.5 System building under KADS

The full KADS life-cycle model is represented in Fig. 2. However, perhaps a simpler and more abstract way of thinking of the system building process is as a *modelling* process in which a real world domain is abstracted and then mapped onto a model of the system at the same level of abstraction as itself. This system model can then be mapped onto a more detailed and less abstract model, which can then be translated into an implemented system. This process is illustrated in Fig. 3. Model M1 is essentially the conceptual model, expressing a description of the desired behaviour of the system resulting from the analysis of the real-world behaviour of experts. M2 is a high-level design modelling M1 in terms of the system architecture. M3 would correspond to a detailed design, in which all the physical blocks, processes, data structures and internal interfaces are described: this may then be easily translated into an implemented system.

Any more detailed description of the work done on the KADS methodology

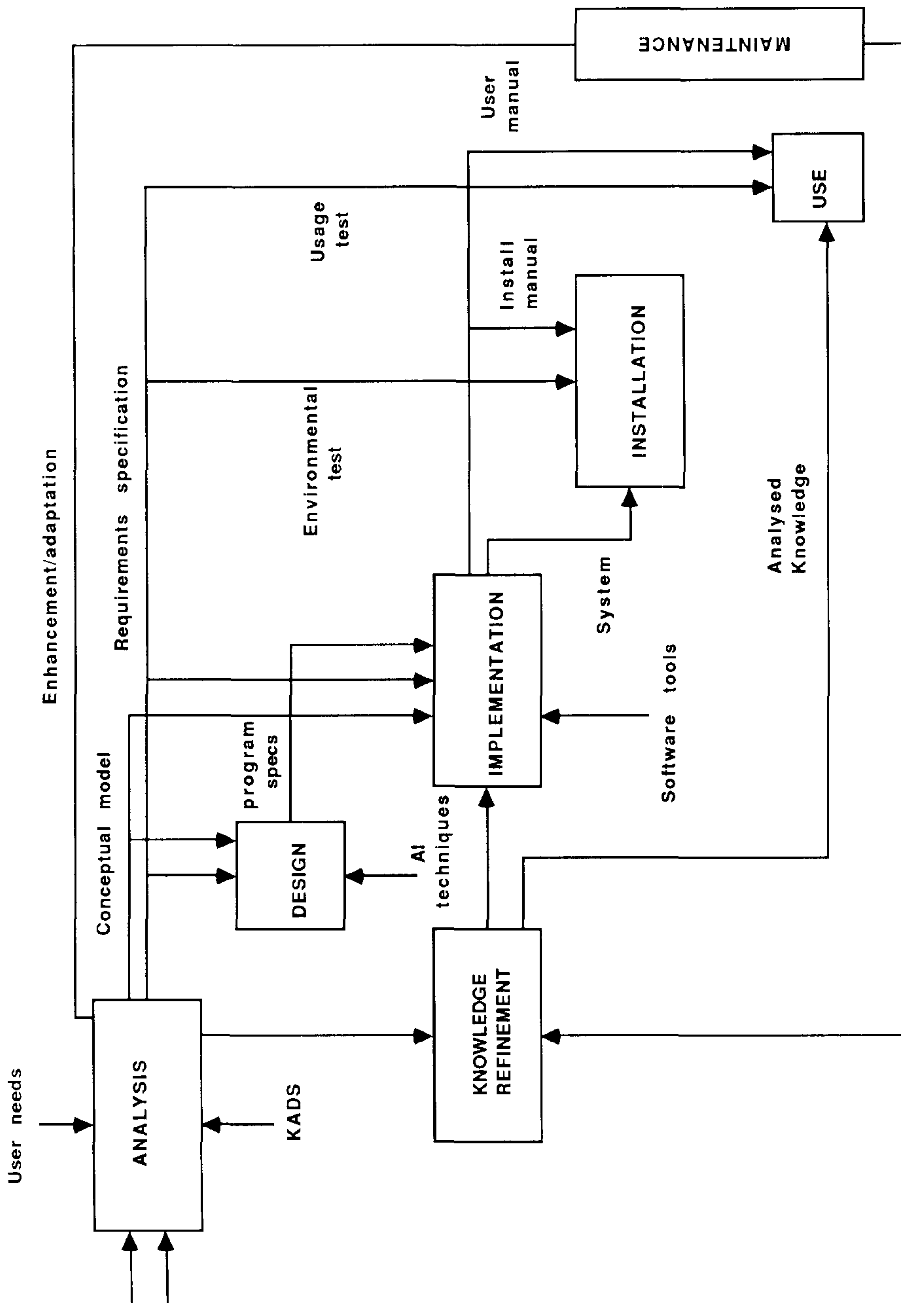


Fig. 2 KADS life Cycle Model

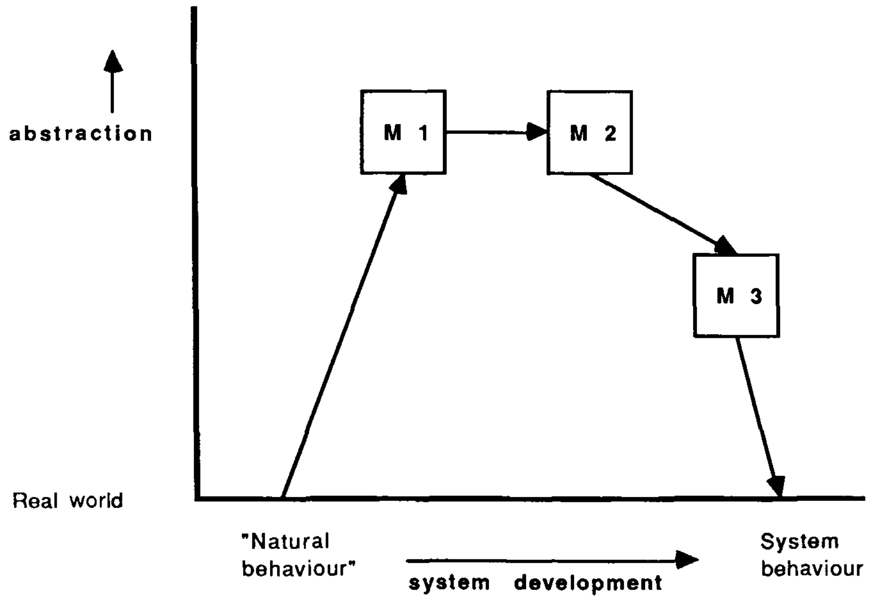


Fig. 3 KADS system Development

would be out of place here: interested readers are directed to the references given in the Bibliography. The remainder of this paper is devoted to an outline of the evolution of the SMIS project to its present state; a high level description of the system's key concepts and design; and an evaluation of the project with some reflections on its implications for future systems.

3 SMIS – the Strategic Marketing Information System

The scoping and analysis phase of the KADS F8 Experiment (SMIS) was accomplished by STC Technology Limited (STL), jointly funded by STL, ICL Retail and the EC through ESPRIT. The design phase, which is now concluding, has been carried out by designers from STL and ICL Retail (the authors), with funding from ICL Retail and ICL Public Services. The project is awaiting a decision on collaboration with a major provider pending the commencement of the detailed design and implementation phases. The principal concern of this paper will be an outline of the design of SMIS, but first a brief account of the earlier phases follows.

3.1 SMIS – scoping

Before any work in this phase started, a detailed set of aims and objectives was agreed with the client – ICL Retail. The agreed aim of the system was, basically, to improve the quality of information available to marketeers, through *efficient* and *effective* access to *relevant* information; and the objectives of the client were to facilitate the sale of ICL hardware and software, obtain a competitive edge over other retail system suppliers and to enhance ICL's image as an innovator by introducing a new generation of out-of-store systems.

Following this, a series of scoping interviews were conducted with ICL marketeers and industry consultants and with an external market research consultancy group. The methodology underlying these interviews was Checkland's Soft Systems Methodology (SSM). The limits of the Strategic Marketing domain were agreed, a high-level functional decomposition of this domain developed, and a set of core functionalities outlined. The phase provided a good introduction to the marketing domain for STL researchers; but it was felt that the use of SSM was only a partial success, largely due to lack of access to end-users and the generic nature of the system.

3.2 SMIS – the analysis phase

This lengthy phase followed the KADS methodology with two streams of analysis – an external stream and an internal stream running in parallel with one another. The fundamental analysis technique was, once again, the structured interview and a set of guidelines and support documents for interviewers was developed. The deliverables were a Requirements Document, estimating the feasibility, scope and organisational requirements for SMIS, and a Model Document, which described the Conceptual Model that had been developed for the marketing domain. The Model Document used KADS techniques to draw up a functional description and decomposition of the proposed system, and offered preliminary and tentative pointers to the translation of the functional specification into a design.

Also delivered were two screen demonstration systems developed using PCE Prolog running under UNIX on SUNs. The first demonstrator illustrated the workings of a generic SMIS system, the second an SMIS system operating over the retail domain of the potential collaborator.

3.3 SMIS – design phase

This phase has concentrated on the development of a design for a pilot SMIS to be developed with a collaborator. Although technically a high-level design, we have found that the work has taken us into realms that might more properly be called low-level design – the distinction is a fairly arbitrary one anyway. The proposed pilot would possess the HCI and the complete system architecture of a full SMIS and would be restricted only in that a limited range of market models would be configured for it, and that it would only address one database (and database type) – a relational database. A set of system tools, proposed in the analysis phase, which would allow the user to configure models and tasks, is also to be omitted from the pilot (although it would be an essential part of any full SMIS).

The remainder of this paper is devoted to a description of this SMIS design and to an evaluation of the potential of such systems.

3.3 SMIS – the design

It is legitimate to view the SMIS system as an *interface* between the marketeer user and a range of data sources. This is not simply to say that the system retrieves data for a user, in the manner of a database management system. More subtly, SMIS allows the user to *specify* tasks expressed in terms of that user's view of the world; *interprets* and *translates* those tasks into a set of operations (of which data-retrieval may only be one) expressed in terms of the system and database view of the world; *retrieves* appropriate data from the appropriate sources (the full SMIS will address a whole range of computerised data sources); and *integrates* and *retranslates* the results of these operations into objects existing in the user's world-view.

Figure 4 illustrates the situation. The marketeer sees the world in terms of market models and processes, market segments etc. The organisation for which the marketeer works will have data available in the form of databases and files, relations, entities, tuples, records, etc. The purpose of SMIS is to mediate between the two views through a series of knowledge based mappings. Some of these key concepts in SMIS are now examined.

3.3.1 Key concepts – marketing

3.3.1.1 Market models, processes and tasks Crucial to an understanding of a marketeer's view of the world is the notion of a *market model.* This is an ill-defined term as it is used by marketeers themselves; perhaps a loose and

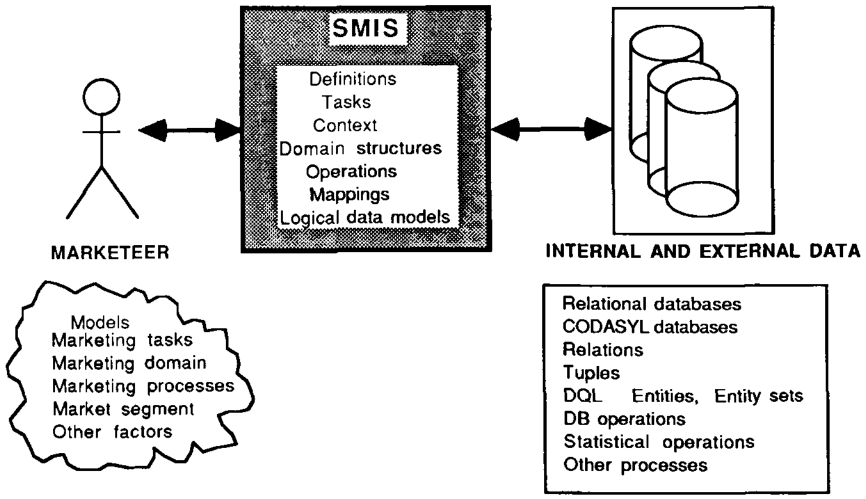


Fig. 4 SMIS – an interface between marketeers and data

unhelpful definition of it would be a structure representing some aspect of a market within a particular context. More rigorously, a market model may be defined as a specific relationship between entities in the marketing domain (marketing domain entities would include such things as product, customer, outlet, etc.) that describes which attributes of the given entities are to be highlighted, compared or contrasted. An example of such a model would be *market share*, in which the entity SALES is related to the entity MARKET-SIZE by a simple function. A *marketing process* may be defined as an activity involving the creation or analysis of a market model. A *marketing task* is some well-defined set of marketing processes.

3.3.1.2 Marketing context Many marketing models are generated within a particular context and only retain their particular meaning against that context. For instance, market share will have different values in the following situations:

Market share	
Market share	*Product P*
Market share	*Product P Region R*

The italicised qualifications attached to the second and third model specifications restrict the range of data that is relevant to the model. Thus, an essential part of the specification of any created market model will be its context.

3.3.1.3 Analyses The creation of market models is a generic marketing task: the subsequent analysis of them is another. Analyses of models may be either *statistical* (eg. discriminant analysis, cluster analysis) or *heuristic* (a more loosely defined group of functions comprising such things as model comparison, model classification, etc). Any marketing information system would be required to support at least some of these analyses.

3.3.1.4 SMIS – functionality The aim of SMIS is to allow the user to extract marketing information from a range of data sources. The emphasis here is on the word *information*. The user is assumed to have no knowledge of the underlying structure of the data sources or of the means the system uses to extract data from those sources. Furthermore, the data, when it is retrieved, is arranged and presented in such a way as to be meaningful to the user in marketing terms – in terms of market models, contexts, etc.

Thus the user specifies the market model s/he requires SMIS to generate in terms of a *keyword* chosen from a range of keywords available in the system. This keyword maps onto a Definition that has been set up and is stored in the system (see 3.3.2.1 below) and represents some marketing task, which may range in complexity from a simple extraction of data (eg. Total Sales for all stores in the SE region) to a complex marketing task (assess a particular

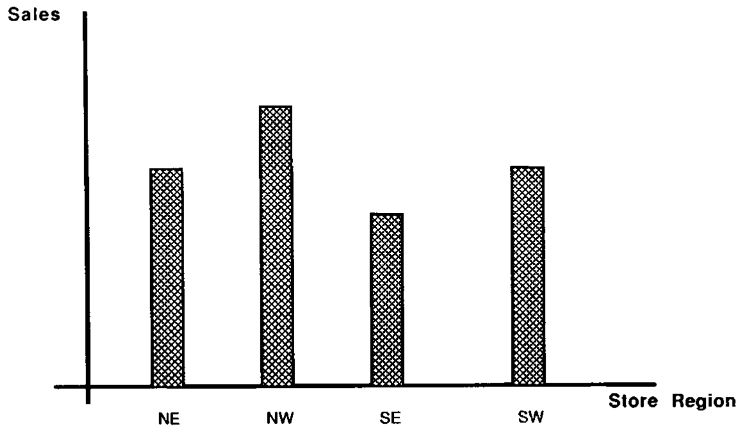


Fig. 5 Example Market Model 1

catchment area for its suitability for the building of a new store). From the point that the keyword is entered onwards, SMIS proceeds without further intervention from the user.

The resulting market model may be expressed in some graphical format. A choice of such formats is available; and a model may be displayed in any suitable format. Two examples are given below: the first (Fig. 5) is the result of a simple data extraction, the second (Fig. 6) is a more sophisticated model, expressing projected sales given known population trends.

3.3.2 The representation of market concepts in SMIS A number of structures are used in SMIS to map onto the generic marketeer's world picture. The most important of these are

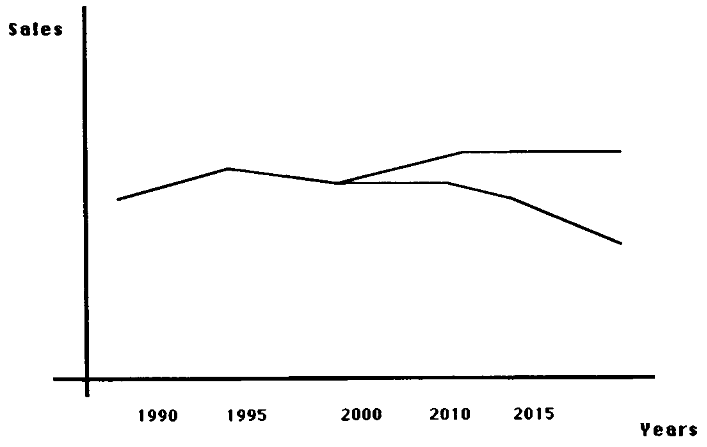


Fig. 6 Example Market Model 2

Definitions
Context
Domain Structures

These structures are the starting point for a mapping process, the end product of which will be some query (or set of queries) to a database, or some analysis of a previously created model, or some complex of model creation, analysis and other operations. A brief account of each of these now follows.

3.3.2.1 SMIS definitions Definitions in SMIS are frame-like objects which encapsulate, declaratively, marketing tasks. A frame – in AI terms – is simply a structured object with a name and a number of 'slots' containing information. The slots of each Definition frame in SMIS contain 'housekeeping' details such as the definition name, help and explanation pointers, but also a specification of the marketing task that the definition is to carry out in terms of *operations* and *parameters*. Operations are functions known to the system and may be of various types: *database operations* (usually data retrieval, but also data retrieval combined with some other numerical or formatting operation performable by the DBMS); *statistical operations*; and *knowledge-based operations* such as 'compare', 'classify' and 'generalise'. The Definition frame also contains a declaration of the Definition's result type, for internal type-checking purposes. Parameters may be *marketing values* stored declaratively in the system (eg. inflation rate), other *models*, other *definitions*, or special frame-like entities, known as *atoms*, that map directly onto datasets retrievable from databases or files.

It is an essential part of the SMIS philosophy that Definitions should be fully configurable and tailorable by the user, that new ones can be created and old ones deleted. A full SMIS will provide intelligent editing functions to make this possible.

3.3.2.2 SMIS context In SMIS, the marketing context is maintained throughout a user session as a dynamic data structure. Initially, it will be set to some default value, but will change throughout the course of the session as the user generates new models and/or alters the context explicitly. When a model is to be generated, the user is invited to leave the context at its current value or to alter it explicitly. It is also possible to configure Definitions that will generate a model with a pre-specified context.

3.3.2.3 Domain structures These data structures have a two-fold purpose. Firstly, they are a declarative representation of the entities and their interrelationships of which the user's view of the domain is composed. This, in most cases, is some hierarchical conception – for instance, the PRODUCTS a retail company sells may comprise a number of PRODUCT-GROUPS, which in turn are made up of a number of PRODUCT-TYPES, etc. However, the relationships within any one hierarchy have proved to be too complex to represent as a simple tree; a more complex data structure is required. Secondly, the Domain Structures represent the mapping of user-view entities onto database entities in the databases (or other storage system) of which SMIS is aware. This is expressed by a mapping from certain nodes in a Domain Structure onto elements of various logical data models – one for each database (or other storage system) that SMIS knows about. The nature and function of the logical data model is described more fully below.

There are a number of Domain Structures, one for each major marketing domain entity. Again, it is part of the SMIS philosophy that these structures should be tailorable.

3.4 SMIS mapping procedure

Thus the marketeer user's view of the SMIS system is expressible entirely in terms with which s/he is familiar: market models and marketing processes (which may be named in any private way the user chooses); more complex marketing tasks; and a marketing domain comprising objects and relationships understood by the marketeer. As described in Section 3.3, the task of SMIS is to map from this view of the world to the computer view. Typically, this is done through a series of mappings and procedures, a simplified version of which is depicted in Fig. 7.

The detailed syntax/structure of each mapping entity has been worked out and is specified in the SMIS Design Document. However, most of these details are omitted, partly for reasons of confidentiality and partly for reasons of space. Below is given an outline account of the nature of each mapping and procedure.

3.4.1 Mapping 1 – Def→IQL Marketeers may inspect any model that has been generated at some previous time by a separate display process. Their access to marketing *processes* is through the invocation of a Definition. A Definition may simply be called by keyword at the HCI and the mapping process is set in train.

The first stage is the processing of the relevant parts of a Definition into a flat list of instructions to the system, expressed in an internal language named IQL (Internal Query Language). The instructions may concern the retrieval of some data, the application of some operation to data or to a previously generated model, etc. Before being interpreted by the next mapping phase, the stream of IQL statements, known as the IQL Statement Set is type checked to prevent an inconsistent set, or statements expressing irrational operations, being passed to the interpreter.

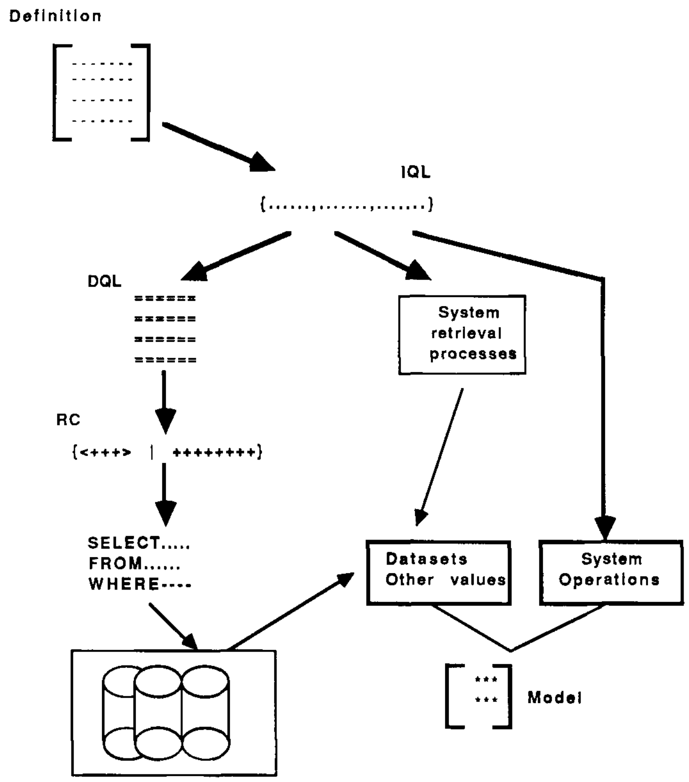


Fig. 7 SMIS – Mappings and Procedures

3.4.2 Mapping 2 – IQL→DQL The processing of IQL statements may, in fact, yield a number of alternative results, depending on the nature of the statement processed. Some statements may be mapped onto system operations (a statistical operation, for instance, such as cluster analysis), others onto requests for data retrieval. Those IQL statements concerning data retrieval are mapped to DQL (Data Query Language).

DQL is an abstract internal language which expresses the data to be retrieved in terms of the databases known to SMIS, database entities and constraints on data. Since a full SMIS will access a number of databases, it is likely that any one Definition may require data items to be collected from more than one, so DQL must be able to express *from which* database an item is to be retrieved. Database entities may be particular relations, columns and tuples in the case of relational databases, or entities, entity sets, owners and members in the case of CODASYL databases. Constraints are introduced by the market context, as described in Section 3.3.1.2 above.

The actual process by which the mapping is achieved is a complex one, which involves the access of Domain Structures and a *logical data model*, a schematic map of each database addressed by SMIS and stored in the SMIS knowledge-base. Readers interested in the details of this mapping process are referred to the Design Document.

3.4.3 Mapping 3 – DQL→RC The DQL statement set is then mapped onto an abstract model of the query that will be required to access the required data from the necessary database. Since the SMIS pilot will only access a single relational database, the mapping language in this case is *Relational Calculus* (RC). Relational calculus is a language based on predicate logic, embodying quantifiers, range variables and logical connectives, which is capable of expressing the full range of queries possible on a relational database. The general form of an RC statement is

$$\{<R_1.A_1, R_2.A_2,\ldots,R_n.A_n>|C\}$$

Where R_i is the name of some set of tuples (relation), A_i is the name of some column and C is the condition representing the defining properties of the set.

3.4.4 Mapping 4 – RC→RQL Relational Calculus is the foundation of a number of real query languages (RQL) such as SQL, QUEL, ILL, etc. The mapping from RC to any of these is straightforward. At the end of this mapping process, data is retrieved from the relevant databases, etc; integrated with other parameters retrieved from elsewhere (for instance, the current inflation rate, the GNP, etc.), and various numerical, statistical or heuristic operations are applied to it, as specified in the original definition (eg. data might be sorted, summed, or have some statistical operation performed on it). The result is known as a *model value*. This is stored and a new model frame created containing 'housekeeping' and reference data, such as the model name, date, help, etc and a pointer to the model value. This, in turn, is stored and the newly created model may be referenced for display, or as an input to some other modelling or interpretation process, at any future time.

3.5 Process and session histories

SMIS keeps records of the processes and tasks it has undertaken and of each session with a particular marketeer. These histories may be displayed and the models that have resulted from the execution of the processes and tasks classified in various ways.

3.6 SMIS HCI

SMIS has a WIMP-based, graphical HCI. Models may be displayed in a number of alternative graphical formats: bar-charts, line-graphs, pie-charts, maps, trees, etc. and the user may switch display between formats as required and as appropriate.

The system provides context-sensitive, on-line help with any item of system functionality, and two varieties of explanation are provided: static and process. *Static explanation* explains the meanings of various constructs in the system, particularly Definitions; *process explanation* indicates to the user how a particular model came about, by what processes it was created.

3.7 SMIS Architecture

The proposed architecture of SMIS may be traced back to the origins of the system in KADS. The system does not have a separate knowledge-base and inference engine, supported by layers of more conventional software, in the manner of traditional expert systems. The system consists of a number of specialised modules, each committed to some highly explicit area of functionality, communicating via a CONTROL module and sharing results and conclusions with other modules via a BLACKBOARD. Certain of these modules correspond roughly to some layer in the KADS four-layer knowledge hierarchy: for instance, there is a module responsible for reasoning with objects in the marketing domain; certain other modules will encapsulate some area of system functionality: for instance, another module specialises in control of and access to the database, another in control of HCI functions. Each module may have a local database and inference capacity and will incorporate an appropriate mix of KB and conventional design and implementation techniques. The overall architectural conception of the system may thus be summarised in the diagram in Fig. 8.

The only communication that modules have with one another is through messages passed through the Control Module (which has no 'intelligence' of its own – it merely oversees the running of the system) and which are represented by thin lines; and through *results* of local operations, which are shared with other modules via the blackboard (thick lines).

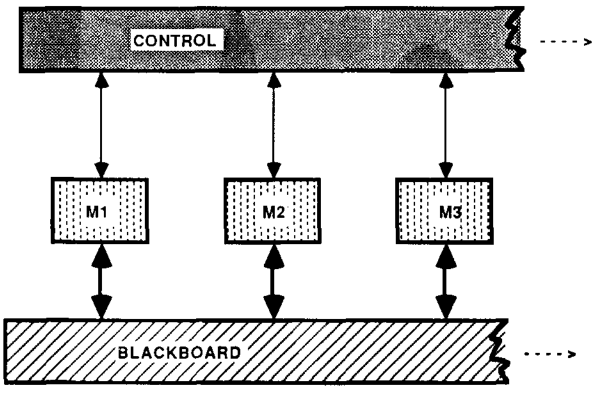


Fig. 8 SMIS Architecture

3.8 SMIS implementation

No firm decisions have yet been taken as to implementation languages, etc. This decision will be taken finally at the detailed design stage. However, it seems likely that the SMIS system will run on a SUN 3 or 4, and that it will be implemented in a mixture of C and Prolog. The pilot system will have an internal relational database, probably INGRES.

4 Evaluation – the future of SMIS and intelligent modelling systems

It has been argued that SMIS is "not a knowledge-based system". If this means that SMIS does not have a separate, monolithic knowledge-base with inference engine, and that the KB is not encoded in the form of production rules, and if the question of what label to apply to the system is deemed to be important at all, then the 'charge' is a true one – SMIS is not a copy of MYCIN, PROSPECTOR or any subsequent rule-based system. However, it is the designers' belief that SMIS is not only a knowledge-based system but also a representative of an important generation of such systems. These systems are embedded in, or in close contact with, existing software (eg. databases, dp systems); they incorporate a mix of conventional and KB techniques; and – once delivered – they are *configurable* by the user organisation, and thus can be changed easily to meet changing demands.

At present, SMIS awaits decisions on industrial collaboration before it can be taken forward to the detailed design and implementation phases. The experience of the authors and the other designers of SMIS with customers all points to a requirement for such systems in industry and commerce. The value of SMIS need not be restricted to any particular sector, such as retail – it would be equally useful in insurance, manufacturing, public services, etc. In fact, any large organisation that maintains large quantities of data and seeks to use it effectively as a source of *information* would have a use for SMIS.

References

The following KADS and SMIS documents give greater detail of the points discussed in this paper – available on request:

BARTHELEMY, EDIN and TOUTAIN: *Requirements Analysis in KBS*, CAP Sogati Innovation, 1987.

SCHREIBER, BREDEWEG, DAVOODI and WIELENGA: *Towards a Design Methodology for KBS*, University of Amsterdam, 1987.

CAMPBELL, CHEESMAN, LEWIS and TANSLEY: *SMIS – F8 System Documentation*, STC Technology Ltd., 1989.

CHEESMAN, DOBBYN, TANSLEY and WALSH: *SMIS – High Level Design Specification*, STC Technology Ltd., 1989.

A Conversational Interface to a Constraint-Satisfaction System

H. Lesan & N. R. Seel
STC Technology Ltd., London Road, Harlow, Essex CM17 9NA

Abstract

Many modelling problems, such as the construction of project plans, spreadsheets etc can be considered as the construction of a term in a formal language, under constraints.

The activities involved in such term-construction, such as expanding an activity into sub-activities, or querying a resource database, may rather naturally be given a hierarchical structure. As Grosz showed, such structures in their turn may induce a structured task-oriented dialogue.

The paper describes the way in which such construction, viewed as a cooperative problem solving activity, gives rise to a dialogue between the (two) participants. A system is described which uses internal task and term models to drive a mixed-initiative dialogue. At each point in the dialogue, the system generates a virtual tree of natural language sentences which constitute responses the user is able to make, including the giving of information, requests to the system for further information or opportunities for the user to move the focus of discourse.

The virtual tree is displayed on a high-resolution screen as initial phrases: the user may explore the tree structure to construct sentences by movement of the mouse, using the technology of "walking menus". A transcript of the dialogue, and a graphical model of the current state of the constructed model are also displayed on the screen.

1. Introduction

In 1987/88, the final phase of the Alvey "Adaptive Intelligent Dialogues" (AID) project developed a number of systems exploring aspects of adaptive user-interface design. The systems were all loosely connected within the framework of a collaborative authoring application, and were implemented in Smalltalk-80.

One such system was the *Task Organiser*, which looked at the problem of supporting a professional worker in the task of constructing a project plan[1]. We wanted particularly to capture the fact that project planning occurs in a sophisticated context of organisational goals, financial constraints, personnel availability, skills etc, and that the planning process is one which is repeatedly interrupted and returned to. This prompted us to take seriously questions of context and process (cf [WF86]), and in the process to reject a straightforward "direct manipulation" solution.

The purpose of this paper is to explain in more detail our contextual, process oriented view of much professional office work by taking the case of project-planning, and the *Task Organiser* as an example. For more information about the *Task Organiser* system architecture, see [WL88].

2. The Project-Planning Example

2.1. An example project plan

Consider an informal representation of a project plan to produce a prototype from a requirements document (Fig. 1). Deliverables are represented by rectangles, activities by ellipses. Each deliverable or activity has a collection of attributes such as start-date, end-date, person responsible for it, etc. It is possible to expand an activity into a sub-plan, as shown in the case of activity **a3** below.

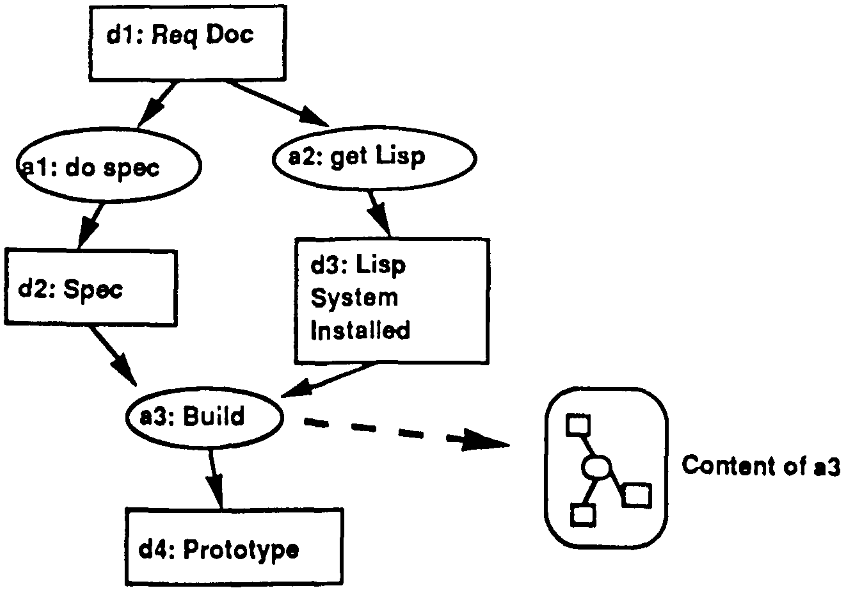


Fig. 1

[1] The AID implementation supported project planning for multi-author document preparation.

2.2. *Project Plans as Syntactic Structures*

We can formalise plans of this sort as terms generated by a grammar, thus:

ProjectPlan : : package(name, start, end, persons, input, output, etc) |
setOf(ProjectPlan | package(name, start, ...))

The operator "package" is a syntax constructor which corresponds to activity. Items such as "name", "start" etc in its argument list are variables, which may be bound to constants denoting activity name, activity start-date etc. The operator "setOf" organises a non-ordered collection of items in the category of its argument.

Well-formed Project Plans are subject to a number of constraints such as:

* start $\leqslant$ end
* persons are available in the period [start ... end]
* person skills are appropriate to input, output
* etc

Writing **PP** for project plans and **p** for packages, two examples of the "shape" of project plans constructed from this kind of grammar are shown in Fig. 2.

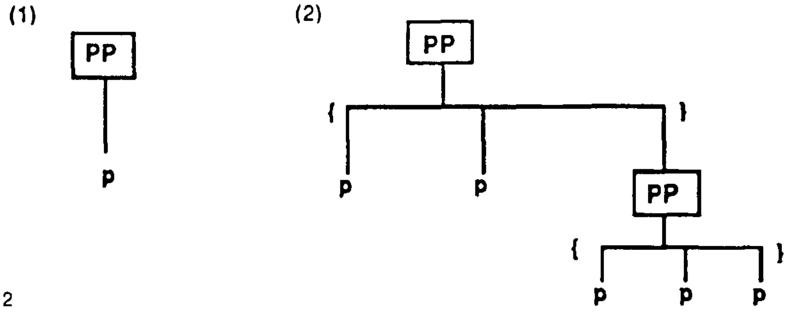


Fig. 2

The construction of such a project plan is a dual-process:

(1) expand a non-terminal node, e.g. Project-Plan $\rightarrow$ a set of packages,
(2) bind the variables in packages to concrete values.

A project-planning support system such as the *Task Organiser* therefore needs to support project-plan and package objects as data, together with operations to expand them syntactically, and to bind variables[2]. In supporting the latter task especially, other resources are also necessary.

[2] Some types of constraints could be represented as attributes in an attribute grammar. A very simple version of the *Task Organiser* could therefore be thought of as a syntax directed editor for an attribute grammar.

2.3. Other System Resources

Creating and maintaining Project Plans cannot be done in isolation. To determine suitable individuals to accomplish activities, there needs to be a database of personnel resources, linking individuals and their skills. Similarly, to determine availability, there needs to be access to diaries. This can be done either directly, via on-line diaries, or indirectly, by asking individuals about their availability by phone or e-mail. Similar considerations apply with respect to the customers of the project, management etc.

The office automation strategy at STL assumes that all these database and communicational resources will be accessible via the computer system. Hence the project planning task is conceptualised within this wider sphere of continuing negotiation: it is not simply a stand-alone "one-off" development of one person's snapshot view.

We are therefore led to the notion of a project planning system, specifiable as a complex system involving project plans, personnel databases, machine-resource databases, personal diaries, deliverable databases etc.

2.3. Constraints and Sequencing

We have already seen how the information necessary to accomplish the project-planning task is distributed between the user, other people, and various databases.

The procedures involved in project-plan creation should also be distributed. As we saw, a project-plan is a heavily constrained term. For example, the start-date of an activity must precede its end-date; the utilisation of any individual across a collection of activities must not exceed 100%; skills must be matched to the nature of inputs and deliverables etc. It is usually not difficult to formalise such constraints, which are essentially constraints over how package variables are to be instantiated, but how should they be implemented?

Although in principle it could be left to the user to internalise such rules, in practice this is a recipe for user-error. A better design decision is to embed the constraints within the computer system, either to check user input, or better, to display to the user the options for instantiating each variable in a package, subject to the constraints as applied to previously instantiated variables.

On this basis the procedure for instantiating variables has been decomposed into two phases: "generate" followed by "test". Generation-under-constraints is allocated to the machine, and selection ("test") allocated to the user (we will mention the technology necessary to accomplish this below).

Although it may be less obvious, it is also useful to distribute responsibility for subtask sequencing. Although in principle it is possible to bind variables in a partially-constructed plan in an arbitrary order, this does not occur in practice. On the contrary, a systematic development of a project plan is the norm, corresponding to breadth/depth-wise expansion and instantiation at the package level (punctuated by interaction with other agencies and resources such as customers, personnel databases etc)[3].

Considering the creation/maintenance of a project-plan as a mini-project in its own right, it seems plausible that a project-plan to construct project-plans would have a systematic character, imposed by the requirement to optimally organise the problem-solving activities necessary to accomplish this task. If we can identify normative models of the process of project-plan creation and maintenance, these may then be embedded in the computer system. This permits the generation of "contexts-of-work trajectories": paths for the user to navigate through the task.

3. Designing the hci

3.1. *Intensional and extensional modes of description*

Direct manipulation (WIMP) interfaces are in some sense extensional, in that signs on the screen are meant to denote individuals directly in some semantic domain (the functions named in menu entries may be considered functional individuals, since in general one only wishes to apply them).

In principle it is always possible to treat a semantic domain set-theoretically, representing individuals by signs such as icons and strings, and indicating set membership by some binary attribute of signs such as inverse video, or spatially, by enclosing signs by grouping-constructs (boxes, folders etc). Individuals and sets can be indicated using these techniques both by the machine, and by users via keyboard and mouse. This works well for applications where the individuals and sets of interest can be easily delineated by mouse/screen/keyboard technology. Consider for example text processing, where cut/paste operations are applied to sets of contiguous strings, which have been previously identified by mouse-dragging.

Where the sets are more complex, as in database enquiry, or the individuals more abstract (consider which individuals are denoted in the sentence "I now want to consider manpower requirements"), then it is more convenient for the user to deploy *intensional descriptions* and let the system sort out the denotation. We take an intensional description to be one whose denotation cannot be computed without taking account of additional contextual values.

[3] Note the analogy between the linking of plan structure and sequencing order here, and the close connection between task and dialogue hierarchies observed by Grosz [Gro77].

3.2. The Task Organiser *Design Rationale*

In the case of the project-planning example, it would be a possible design option to treat a (partially constructed) project-plan as a displayable graph-like object, as in Fig. 1 above, to support editing operations to set node and link attributes, and to establish new links and nodes, in the style of hypermedia. Provision would have to be made for access to other resources, in the planning system, such as communications with colleagues, database query etc. This too could no doubt be coerced into a WIMP format.

In such an hci design, the project-plan artifact is considered primary, while the process of constructing it—assembling information, deciding what to do next—is only *implicit* in the history of the user's navigational and selectional choices. We hypothesise however that for professionals such as project-plan constructors, the elaboration of the *process* is often of primary importance (e.g. because such people often get interrupted, and forget the strategies they were working to; or wish to leave a record of design decisions for colleagues). We therefore decided to make explicit representation of process a priority.

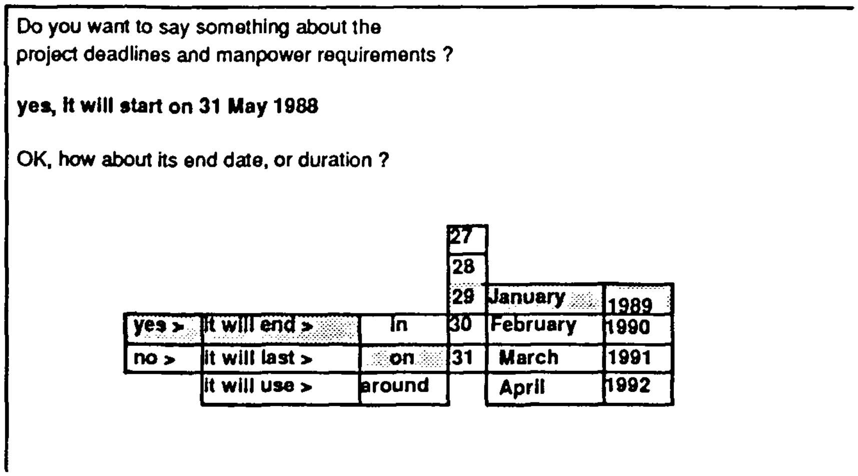
As discussed above (in **2.3**), the process of project-plan creation is highly contextual, involving database enquiries, interrogative and confirmatory dialogues with colleagues, and plan-elaboration decisions made in the context of the development of the plan *at that point*. Although possible, the representation of design history and design options extensionally, in terms of "snapshots" of previous and potential system states, is unacceptably clumsy. We decided instead to present to the user intensional descriptions of the actions it was appropriate to carry out, using natural language sentences, and to preserve the history of the resulting dialogue as a transcript.

3.3. The system we built

At a certain point in a project-planning task the *Task Organiser* screen might look like Fig. 3 below. Here, the user has already volunteered the start date of the project (bold type), and has now been asked by the system to elaborate on the end date (plain type).

Opposite "yes" in the left hand menu is an arrow head > which indicates a further menu. In this way a branching structure of possible sentence continuations is made available. Menus such as the three right hand side ones, giving date information, are scrollable. The domain of values presented is sensitive to the constraint that the end-date of a project is after the start date, hence no values prior to **31 May 1988** are presented.

The user selects the required sentence by moving the mouse to the right, scrolling where necessary, and by finally confirming the selection. The selected sentence is then added to the transcript, the action effected by the sentence is carried out, and the system responds with a reply and a new menu choice.


Do you want to say something about the
project deadlines and manpower requirements ?

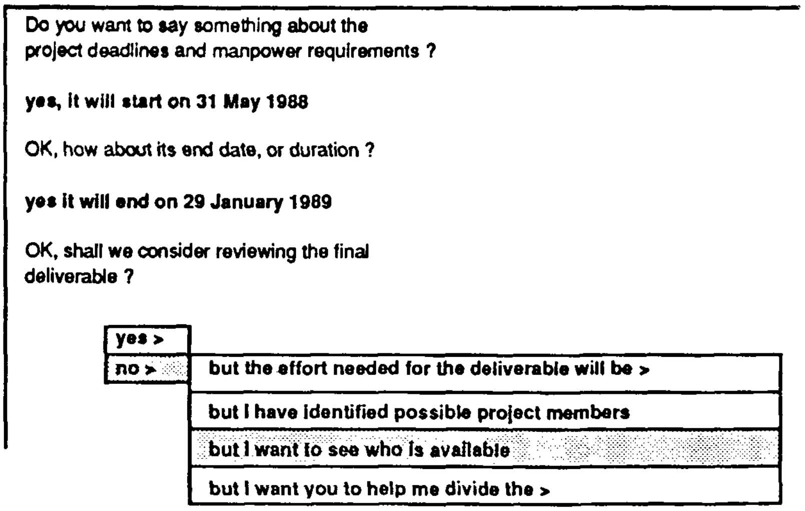
yes, it will start on 31 May 1988

OK, how about its end date, or duration ?

Fig. 3

Note that the user could have replied "no", and gone on to address some other issue, as we see below, a little later in the dialogue, in Fig. 4.

The transcript shows the process by which the user "got to here": the menu-sentences invite further actions from the user in a process-oriented language. It is possible to combine the direct manipulation and dialogue paradigms by showing a visual representation of the plan so far in another pane on the screen. Something like this was briefly explored in our prototype.


Do you want to say something about the
project deadlines and manpower requirements ?

yes, it will start on 31 May 1988

OK, how about its end date, or duration ?

yes it will end on 29 January 1989

OK, shall we consider reviewing the final
deliverable ?

yes >
no >
but the effort needed for the deliverable will be >
but I have identified possible project members
but I want to see who is available
but I want you to help me divide the >

Fig. 4

3.4. *Is it dialogue?*

At first sight, this "conversational" style of interaction looks remarkably like human dialogue. The user and the machine system are both generating utterances, which seem best understood in the framework of speech-act theory, as illocutionary acts such as commanding, requesting, and asserting.

It might be argued that both participants are violating the Gricean conditions, by which they ought to be intending that their conversation partner recognise their intensions [Gri57,69; Sea69]. Stripped of philosophical concerns about the nature of meaning, the main function of the Gricean conditions however is to permit each conversation partner to predict the cognitive situation and consequent behaviour of the other.

In the case of the dialogue model in the *Task Organiser*, the analysis of user beliefs and intentions at various points in the task was done by the *Task Organiser* designers at design-time. The *Task Organiser* certainly incorporates procedures for package-handling based on cooperative principles in dialogue[4], but it does not model the beliefs or intentions of the user at interaction-time *explicitly*. It is an implementation, a performance model, built in accordance with a normative competence model of what we, the designers, anticipated a professional user and a cooperative assistant system would believe and intend at various points in the task. Our suggestion is that for a stereotyped process such as project-plan creation, such a fixed intentional structure is adequate.

4. Conclusions

The technology of natural language phrases selected and composed by the user is a useful addition to the hci designer's toolkit. Using walking menus and a mouse, techniques of direct manipulation have been harnessed to the power of intensional description. This permits sentences to be input at speed, without the user-overheads of typing and the computational difficulties involved in parsing unrestricted natural language text[5]. We found two kinds of issues emerging in integrating this style of hci with application functionality: those of architecture and interaction structure.

4.1. *Architecture*

A procedural implementation of this hci-style simply associates a sentence-processing procedure with each package-type, binding variables from sen-

[4] In a fashion similar to the automated advisor for train information of [AP80].
[5] Another project which has found this technology useful is the Alvey DSS demonstrator project, which has a similar interface to its Welfare Rights *Advice System* [FL89]. See also the pioneering work at Texas Instruments in the early eighties, described in [TSTM83], [Tho83].

tence-references at need. A more elegant solution would be to define a semantics at phrase level (perhaps as higher-order functions in the style of Montague [DWP85]), and compose the meaning of the sentence from the functionals associated with the phrase constituents of the sentence selected.

If this style of hci is used in connection with a constraint-based problem (as in the example above), then a local change may have global consequence propagated by the constraints. Some form of Truth-Maintenance System is then required to obviate re-entry of all values.

Finally, since the user has some freedom to sequence outstanding activities, the system maintains a (rather simple) "dialogue state model", which it uses to suggest to the user new topics. In particular it ensures that deferred topics are eventually brought back into the dialogue. This brings us back to a consideration of interaction structuring.

4.2. Interaction Structuring

Although human-computer dialogues are not "like" at least some human-human dialogues, they can still be completely appropriate for the task in hand. As we move into more flexible, mixed-initiative type dialogues however, we find that interaction becomes less stereotyped, and that information contingently available in the discourse situation, especially information about the beliefs and intentions of the dialogue participants, becomes a causal factor in driving the dialogue.

The *Task Organiser* type of system described here relies upon the task being sufficiently stereotyped that the cognitive content of the discourse can be determined at design-time, and therefore appropriate conversation routines designed-in. At best this is a crude assumption, and it is hard to see how even the levels of cooperativeness in Allen and Perrault's train information model [AP80] could be achieved without deferring estimates of the conversation partner's intentions until the conversation itself. It is therefore to be hoped that *Task Organiser*-like systems can be an implementation route for some of the very interesting work currently being done in speech act and conversation theory.

Acknowledgements

This research was supported by ESPRIT II project 2105, MULTIWORKS. The building of the first exemplar was supported by Alvey project MMI/006 "Adaptive Intelligent Dialogues".

The authors would like to acknowledge the efforts of Albert Wong, who undertook much of the detailed design and implementation of the AID Task Organiser system.

This paper was presented to an IEE Colloquium on Formal Methods held in December 1989 and is published here with the permisssion of the IEE.

References

[AP80] ALLEN, J.F. & PERRAULT, C.R. "Analysing Intention in Utterances". Artificial Intelligence 15, pp. 143–178. 1980.

[DWP85] DOWTY, D.R., WALL, R.E., PETER, S. & REIDEL, D. Introduction to Montague Semantics. 1985.

[FL89] FROHLICH, D.M. & LUFF, P. "Conversational Resources for Situated Action". In Proc. CHI-89, pp. 253–258. 1989.

[Gri57] GRICE, P. "Meaning". Philosophical Review, vol. 66, 1957, pp. 377–388.

[Gri69] GRICE, P. "Utterer's Meaning and Intentions". Philosophical Review, April 1969.

[Gro77] GROSZ, B.G. "The Representation and Use of Focus in Dialog Understanding". Dissertation, University of California at Berkeley, 1977.

[Sea69] SEARLE, J.R. Speech acts: An essay in the philosophy of language. C.U.P. 1969.

[Tho83] THOMPSON, C.W. "Building Usable Menu-based Natural Language Interfaces to Databases". In Proceedings of the 9th VLDB Conference, Florence, Italy, October 1983.

[TSTM83] TENNANT, H.R., SAENZ, R.M., THOMPSON, C.W. & MILLER, J.R. "Menu-based Natural Language Understanding". In Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, pp. 151–158, Cambridge, Mass., June 1983.

[WF86] WINOGRAD, T. & FLORES, F. Understanding Computers and Cognition. Addison-Wesley 1986.

[WL88] WONG, A. & LESAN, H. 1988. "Task Organiser". Unpublished AID document aid/3/report/s0234.1. (Available from the authors).

"SODA": The ICL Interface for ODA Document Access

Michael J. Coon
PODA Project, CIA, ICL Bracknell

Abstract

This paper introduces the international standard "Office Document Architecture" (ODA), and the ICL PODA project which was crucial in formulating it and is piloting its exploitation. The project demonstrates the interchange of documents between computer-based systems using ODA.

The paper explains why the project developed a storage manager for manipulating ODA documents, and an Application Program Interface (API) for it, with commercial toolkits in support of this interface.

It explores the implications of having a published (probably standardised or public domain as well) interface for creating and manipulating ODA documents, with toolkits supporting them. The consequence is that the interface does not imply a particular format for stored documents, but that the toolkit does.

1 Introduction

This article describes demonstrations of electronic document interchange to place the international standard "ODA" in context. An overview of Office Document Architecture is presented to show how documents are constructed.

The need for a random-access form of stored documents is explained, and also how this leads to a requirement for an Application Program Interface to manipulate these documents.

Finally, the article presents the ICL solution "SODA", and the ODA toolkit that it supports. Current and potential uses of the toolkit are described.

At the end of the article is a Summary, a Glossary and list of references.

2 The PODA project demonstrations

In each of the last three years, ICL has demonstrated, at the CeBIT fair in Hannover, the interchange of electronic documents with other European companies.

If the documents interchanged had been merely memos, then the technology involved would have been no different from that used for demonstrations of electronic mail. The international standard for electronic mail, X.400, is in routine use for sending memos of a few paragraphs both within and between organisations. Many different manufacturers offer X.400, and this form of mail can be sent by utilising several data interchange services.

The simple memo is sufficient for the exchange of unstructured information. However it exploits only the most basic capabilities of modern word processors. The state of the art products in this field can cope with text which is emboldened, underlined, italicised and so forth. At the level of document structure, they offer automatic numbering of paragraphs and pages, footnotes, headers and footers, tabular multi-column layout, and illustrations. None of these features can be included in a "simple memo", but were shown in the demonstrations.

The PODA project was set up, initially in 1986, to pilot the use of a new standard, Office Document Architecture, for interchanging electronic documents having advanced features such as those listed. The name "PODA" is an acronym for Piloting Office Document Architecture. It is now project number 2374 in the EEC's ESPRIT-II initiative to promote information technology in Europe. As well as ICL, there are nine other European organisations collaborating in the project, from five countries.

Office (or "Open") Document Architecture is a new international standard (published as ISO 8613, CCITT T.410 and also as ECMA 101) for the interchange of electronic documents. It lays down a format in which the documents are interchanged, called Office Document Interchange Format, so the complete standard is often known as ODA/ODIF. It is part of the set of standards called OSI, Open Systems Interconnection. Because the PODA project is an international one, it has concentrated on exchanging documents via X.400 mail, but exchange using floppy discs is possible and has also been done.

It is an aim of the PODA project to encourage the transfer of ODA technology into the world of electronic documents. Complex documents, for instance chapters of a multi-lingual report, could be written by several authors in different countries on wordprocessors having national features. The separate contributions could then be sent to an executive editor who assembles them into the final document. All of the structural information would be maintained, and none of it would have to be typed twice. Where necessary, the layout would be readily modified to give a consistent appearance.

The companies in the PODA consortium do not use ODA/ODIF only for demonstrations. Each company has the facility permanently set up, and it is in use for technical communications.

3 Outline of office document architecture

The ODA Standard has been specified to permit the interchange of electronic documents between computer-based documentation systems. In each interchange, the organisation of document content into chapters, paragraphs, footnotes and so on is kept separate from its division into lines, columns and pages.

Accordingly, depending upon how it is used, the standard can permit each recipient who has suitable equipment to reproduce documents in the form intended by the originator. Alternatively, recipients can be enabled to reproduce documents in a modified format, or even to edit them while still maintaining consistency with the original. Keeping the original format is called retaining layout structure and creating a new format, retaining logical structure.

ODA enables documents containing any mixture of character text, raster graphic (facsimile) images and geometric graphics to be encoded and interchanged electronically. The semantics and syntax of these content types has been carefully separated from that of the document structure. This helps to keep the structure simple. It also makes it possible to extend the standard to allow for further content types and encodings in the future without major disruption.

In order to achieve these objectives ODA specifies an abstract architecture for the representation of the information contained in a document. ODA describes the logical and the layout structure of the document, each distinct from its content. The logical structure shows how the document content relates to such items as paragraphs, headers and diagrams. The layout structure describes the physical layout of the document into pages, columns and so forth. A defined layout process connects the two by specifying how the desired layout structure may be generated from the logical structure.

As well as the logical and layout structures of the document, ODA includes the rules for generating and amending these specific structures, that is for specifying all the valid variations of them. These rules are contained in generic logical and generic layout structures, which provide editing guidelines and constrain the layout process respectively. The two generic structures taken together comprise the document class.

More prosaically, the elements of the generic structures provide a way to specify only once those common attributes which recur throughout the document. For instance the way in which the bulleted items in a list is shown will usually be consistent in all chapters. This may conveniently be specified in the generic structure rather than repeated each time bulleted items appear. One result is a more compact encoding and therefore faster and probably cheaper interchange. The other result is that, when extending a document, a template is provided so that new bullets look like the existing ones.

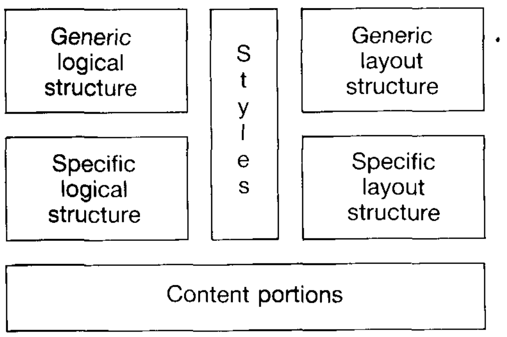

Generic logical structure
Specific logical structure
Styles
Generic layout structure
Specific layout structure
Content portions

Fig. 1 Diagram of FPDA

These four major divisions (generic and specific, logical and layout) of an ODA document are linked by two sets of elements called styles. Layout styles provide the parameters for the layout process required for a given document, equivalent to the required mapping of logical entities onto layout entities. Presentation styles govern the appearance of regions of content. The provision of these two sets of constituents helps in segregating the information in the document and in avoiding unnecessary repetition.

The content of the document is organised into content portions. These are attached to constituents of both the logical and layout structures to show the parts of the document to which they belong.

The diagram, Fig. 1, illustrates all of the above subdivisions.

The diagram shows (with some simplification) how the parts of an ODA document relate to each other. Where all types of constituent are present the document is in "formatted/processable architecture" because the presence of the layout structure indicates that the document has been formatted. Similarly, the presence of the logical structure means that the document could be processed, for instance formatted again with or without having been edited.

Reduced forms of ODA document are recognised as distinct architectures by the standard. These are

- Processable Document Architecture,
- Formatted Document Architecture and
- Formatted-Processable Document Architecture.

The formatted form is used where there is no requirement for editing or formatting, for instance when sending to a printer. Processable form is used where it is expected that formatting will be done, if required, by the recipient.

In the case of processable document architecture only the specific layout

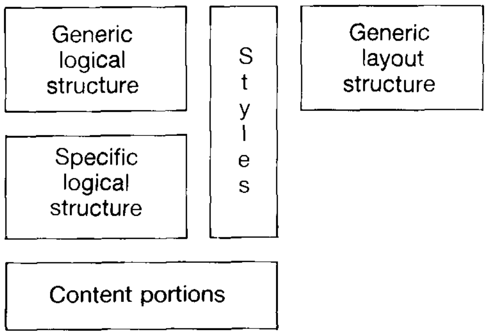


Fig. 2 Diagram of PDA

structure is omitted. This is because it can be regenerated using the rules in the generic layout structure, the layout styles and of course the logical structures. The corresponding diagram, Fig. 2, shows this.

In the case of formatted document architecture both of the logical structures are omitted. Layout styles are also not required, but presentation styles may be present. The corresponding diagram, Fig. 3, illustrates this.

Aspects of both the structuring information and the content in a document can appear many times throughout the document. ODA embraces the concept of "classes"; these are objects (or elements) within the document structure which have the same characteristics and are classified into groups. An object class is a specification of the set of characteristics that are common to its members. The complete set of object classes in the document is in fact the generic structure, also called the document class. The concept of object classes simplifies the creation of the document, improves document transmission efficiency and preserves the integrity of the document structure through editing sessions.

The document content can be of several different types. Those currently

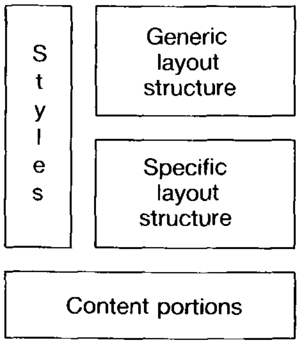


Fig. 3 Diagram of FDA

covered by the standard are text, raster and geometric graphics. Each content type has its own defined content architecture which specifies content structures, coding schemes etc. An important feature of content architectures is their independence. This enables new content architectures to be included easily.

4 How to interchange ODA documents

Authors and typists who are using computer-based systems today to produce electronic documents do not know ODA, nor do they produce ODA documents. Nevertheless it is feasible to convert their documents into ODA for exchange with other systems, and then back again into familiar word-processing formats. This is how the PODA pilot has operated. Only the writer of the converter programs has to understand ODA.

The ODA standard specifies an interchange format for passing documents between users. This format employs an international standard for the specification of the exchange of data in an implementation-independent way, Abstract Syntax Notation One, or ASN.1.

ASN.1 incorporates a formal language for specifying the required syntax (the notation), and for including within this formal specification some information about the detailed encoding. This formal language is itself machine processable, and tools exist for performing this processing.

The ODA interchange format ODIF is specified in the standard in ASN.1, and machine readable copies have been processed using the tools mentioned above. The tool in use in the ICL PODA project is capable of generating programs which can read and write ODIF. For more detail of this aspect, refer to the sections on the toolkit below.

5 Document Application Profiles (DAP's)

For existing document processing systems to interwork using ODA, it is necessary to have Document Application Profiles which standardise mappings to ODA of the conventional document features typically embodied within, and manipulated by, such systems. It is also necessary to have conversion software to translate between the proprietary document representations which are in use and the international standard.

The ODA standard, recognising the complexity of ODA, specifies how to form subsets of the total set of features and facilities for use by implementations which service different levels of user requirements. These subsets are termed "Document Application Profiles" referred to later as DAP's.

Initially the standards organisations intended to define document application profiles. However, work by the PODA project found that application profiles were not sufficient to describe how existing equipment could support and conform to ODA.

Special technical considerations are required for the use of ODA for interworking between products designed independently of ODA. Although the products make their own particular style of realisation for features such as page headings and footnotes, ODA has a powerful and generalised architecture for describing the properties underlying those features. The features themselves are not all identified in the standard.

To solve this problem the concept of document "superclasses" was conceived by PODA. A superclass is the term used to specify how, for an ODA document application profile, the users' perceived features, such as those mentioned above, are encoded in ODA using structures and attribute values.

The concept of a superclass has been adopted by the various standards bodies defining functional profiles for ODA – first the European Standards and Applications Group (SPAG) and then CCITT, and Japanese and American standards bodies. As a result the superclass concept has been incorporated into the ISO/CCITT rules for the definition of document application profiles for the use of ODA.

6 ODA application program interface

Although ISO 8613 defines an abstract architecture and datastream encoding suitable for interchange of electronic documents, it does not define a storage format. The serial form which is defined for interchange is not ideal for storage, and even less so for manipulation.

The ESPRIT project PODA recognised the benefit of an intermediate canonical format to be used in the conversion between the collaborators' systems. Since there are many possible document representations in use, no relationship can be assumed to hold between the ordering of information in ODA (ODIF) and any other document representation. It was therefore apparent that a random access storage manager was needed for the partial results of the document conversion process. Furthermore it was natural to model the partial results information on ODA – which is the common format in the conversions.

Hence one common requirement for random access to information representing ODA constituents was identified and a solution has been implemented.

In order to write programs which manipulate the stored document it was necessary to specify an interface to the storage manager. This is the Application Program Interface or API. Our interface is called "SODA" (Stored ODA). The storage manager which supports it has the acronym "ODASM". Both are in use by ten companies at the time of writing.

SODA has been designed and used to support applications other than conversion. For example, the ODA layout process and imaging to video screens have been implemented.

7 Higher levels of API

SODA is very basic and general. For example it gives direct access to the attribute values specified for a constituent. It does not constrain the values that the attribute may take. Similarly, it was designed as a non-defensive interface, that is, with little error checking, to support trusted system code. The intention was to minimise performance overheads whilst maximising portability over operating systems.

It has been envisaged that a higher level interface should exist above SODA. Such an interface would be (more) defensive for applications use; it would deal in DAP objects such as passage and footnote and would give more convenient functions such as following the textual sequence, performing searches and so on. The interface semantics should be defined with a good degree of separation from language bindings. Thus it should be possible to map such an abstract specification of the interface onto several programming languages as may be required in different development environments. Figure 4 shows this diagrammatically.

It is probably not appropriate to attempt to construct a single interface which would satisfy all possible ODA applications. For instance many search and interrogation functions will be significant only for interactive usage. Other functions of an API may apply the constraints and use the constructs that are specified in DAPs.

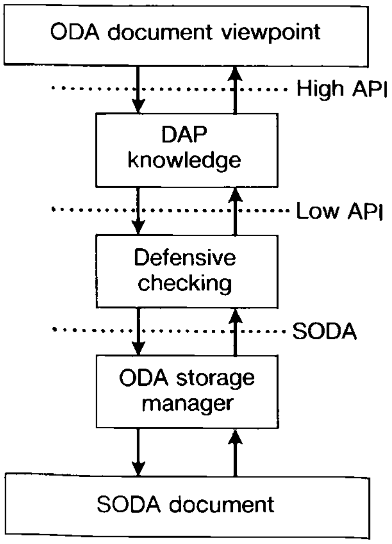


Fig. 4 Diagram of Layered API

8 Local document interchange

For ODA there is an implementation-independent format (ASN.1) for interchange of documents. However this format is not ideal for use as a storage format without augmentation because it is serial with no built-in mechanisms for the fast location of constituents. It is also bulky because most constituents and their attributes are self-defining and multi-layered.

A storage format, on the other hand, should be more compact because it can rely on implementational details (for example, that the compiler has data typing information so that fewer, or no, tags are needed). It may also be bulkier because it holds redundant information to optimise access. For instance it may have indexes to allow random access and also indirection mechanisms to facilitate expansion and contraction during manipulation. The final comparison of bulk depends on the balance between these effects.

It is possible that some storage formats may be used for non-standard document interchange. However, while an interchange format may be used for local storage with disadvantages only of degree, a local storage format which contains implementation-dependent features may be inappropriate for interchange outside its home environment.

9 Program portability

In the present state of technology, most application programs no longer run without support from distinct layers of separate software, even in personal computers. For example, usually at least there is an operating system to control the hardware, and the visible interface may be mediated by windowing or presentation software.

Thus to the conventional measures of portability as specified by X/Open and other groups is added the repertoire of all of the "Application Program Interfaces" (here used not just for ODA but more generally) that are employed by the application.

10 An API and its toolkit

An API which supports the manipulation of a document store may be implemented in the form of a toolkit. This is a library of routines together with both human readable and machine readable definitions of those routines and their parameters.

The toolkit may be implemented for different environments, using the same API and language. If an application is portable through such guidelines as X/Open, then it can remain portable through using the portable API.

A single toolkit in a given environment should generate its document store in such a way that documents can be passed between applications so that those applications form a co-operative suite. This is the case for SODA/ODASM, but might not be the case where a toolkit works only in virtual store, for instance.

It is possible that the same API could have different toolkits supporting it. Within the same environment, such toolkits would presumably be competing and separately sourced. In different environments, the toolkits could of course also be separately sourced, but might also be differently optimised versions from the same source, tailored for that environment. The phrase "same API" is intended to imply that the similarity would be so precise that programs could be constructed with any of the toolkits. Whether the user has a choice of which toolkit is used would depend on how the application was assembled and delivered.

It would be improbable that the document stores that were generated and maintained by different toolkits, even if they were employing the same API, would be compatible so as to allow the applications to interwork to form a suite. To allow this, either use of the same toolkit, or of a standard interchange form such as ODIF, is required.

However, if a set of applications is using the same API, and language bindings, then it should be possible to build all of these applications using the same toolkit. Turning such applications into a co-operating suite should then be trivial. It may involve only making provision for supplying input documents in the permanent storage form of the toolkit, such as an ODASM file. It may be necessary to have access to the source code to be able to do this. Creating such suites of programs must be the primary benefit to systems integrators of the existence of only a very few, ideally one, widely available and utilised API.

A toolkit may usefully contain more than the storage manager. There may be generally applicable applications that may be bundled with the basic toolkit and marketed together. Examples will be found under the heading "ODA converters" in this paper, and formatters may be included as well. (In ODA, formatters implement the line-breaking algorithm, which is natural-language sensitive. Thus there is not just one definitive formatting process.)

11 Potential applications

The other sections of this paper have referred to "applications" without discussing their purpose and capabilities, or have mentioned word processors in particular. Word processors are of course the most common examples of document processors, but it is worthwhile to list some other possibilities.

Note that there is no implication that these particular collections of features should be assembled together into single programs, or that programs having other descriptions might not, working together, provide the same facilities. The following list of applications is intended only to be a guide to areas of relevance.

11.1 Search and retrieval

Given that documents are held in a form which retains their structure and processability, this has two implications for the provision of retrieval-by-content. Firstly a search program has to understand the document storage structure in order to be able to distinguish the content from structural information. Secondly the structural information can be used to refine the search itself, for example to search section headings only, to ignore footnote text, etc.

This application has some processing in common with index-generating applications.

11.2 Forms design

The two extreme varieties that this application may take are a purely interactive program which constructs forms using the logic capabilities of the user, and an automatic one which uses a data dictionary to provide the raw information about fields. In practice, both, or a mixed approach, will be needed.

11.3 Report generators

Any program that generates documents with significant structure is a candidate for generating ODA documents using an ODA API. This includes graphical reports from business graphic generators as well as text.

The automatically generated reports may then be combined with word processed text, other diagrams etc., to form complete documents. This is facilitated by the use of a consistent format.

A distinct and important class of report generator is the form letter generator. Form letters are different from other types of report only in having the appearance of a word processed letter.

11.4 Index generators

Index generators are a variety of simple report generator which uses a document as its input database. The index may be stored with the document (as a distinct part of the content which should not subsequently be indexed again), or may be retained separately.

11.5 Converters

The existence of converters has been assumed elsewhere in this paper. They may be present explicitly for users to invoke when moving files between incompatible applications, or may be embedded in other applications such as mailers.

There are two varieties of converters: those between ODA and other formats, and those within the ODA definition.

11.5.1 Format converters These are the converters to which reference has been made elsewhere in this paper. They convert between ODA and other standard or proprietary formats.

11.5.2 ODA converters There are two standardised interchange formats for ODA: ODIF and ODL. The former has been introduced already, and the latter employs "Standard Generalised Markup Language" (SGML) instead of ASN.1. (ODL is not specified for X.400 electronic mail.) Therefore there are potentially four converters which together deal with the acceptance and generation of both of these formats.

Within ODA there are three "document architecture classes": "processable" (PDA), "formatted" (FDA) and "formatted-processable" (FPDA) architectures. (Refer to Section 3 for more detail.)

The formatting process converts a document in PDA to one in FPDA, and thus this conversion is not included under this heading. Automatic conversion from FDA to PDA or FPDA is not possible because information is missing which would be required for this transition. The other two conversions, FPDA to PDA and FPDA to FDA, are comparatively simple processes.

11.6 Mailers

These applications may not generally be considered as document-handling applications, but that is their purpose. They may be indifferent to the content of a document, but are generally not indifferent to the document format.

11.7 Printers/Print Servers

In the world of the paperless office this category of application is not the final destination of all documents, but is still highly significant.

A print server may include a formatter, or the functions may be separate. This influences the document formats that may be handled (in ODA terms, PDA versus FPDA versus FDA).

11.8 Formatters

This class of applications renders the logical structure and content of a document in a form appropriate to display and or printing.

In practice, the functionality may be included in a print server or mailer. It is not likely to be invoked explicitly by a user except in order to impose a layout that was not anticipated when the document was constructed. For instance, a reformatting may be requested in order to reformat for landscape rather than portrait presentation.

12 A flavour of SODA

12.1 Purpose of application

The PODA project has a permanent facility for the collaborators to exchange documents using ODIF over X.400 connections. Most partners then view, edit and print these documents using their proprietary word processors. Thus each of those partners needed to implement a converter between ODIF and the format used by their word processors. Only the partners who work in native ODA do not need such converters.

12.1.1 Software environment A document converter is the software to transfer a whole document represented in ODIF data stream format to and from one of the proprietary document formats used for a particular manufacturer's word processors. In the demonstrations that have been mounted by the PODA project team the conversion has been initiated by the standard Officepower X.400 mail software.

For incoming mail the flags in the X.400 protocol are interrogated to determine whether ODIF has been received and that conversion is therefore required. For outgoing mail, the system needs to avoid sending ODIF to recipients who would be unable to make use of it. To do this, it maintains indicators of the capabilities of all known possible addressees. These indicators are part of the system's database of the addressing information for the addressees. By this means, it is possible to send a document to a mixture of ODA-equipped and naive correspondents. The latter will get a fallback version of the document in simple text form.

The following architecture shows how such a converter may be structured around the SODA interface. (In this diagram the arrows indicate flow of information.)

The software architecture consists of, starting with a document to be processed in the format for word processor A and with the converted document in the format for word processor B, the following components:

- Proprietary Format Externaliser;
- ODA Storage Manager;
- ODIF Generator;
- X.400 mail network;
- ODIF Analyser;
- Proprietary Format Internaliser.

Of these software components the ODA Storage Manager, the ODIF Generator and the ODIF Analyser were all written in the 'C' programming language and were ported to each of the partners machines and operating systems.

In addition to these major software components other components, such as utilities to print the stored ODA document in human readable form were developed to aid in testing the system.

The ODA Storage Manager was designed by ICL and was ported to both Unix and MS-DOS systems and used by the other project partners in their components of the system. The ODA storage format, SODA, was also designed by ICL.

Use of this architecture for document conversion has the following benefits.

> Development cost is reduced because it is necessary to implement only the particular programs intimately concerned with the proprietary format (the "internalisation" and "externalisation" software). The rest of the programs needed are available as the toolkit.
>
> The portability, performance and resilience of the toolkit has already been established in various environments.
>
> Testing of the new proprietary code is facilitated. There are utilities to inspect the SODA produced, and because SODA is a form of ODA one can thereby confirm that the software for externalisation produces

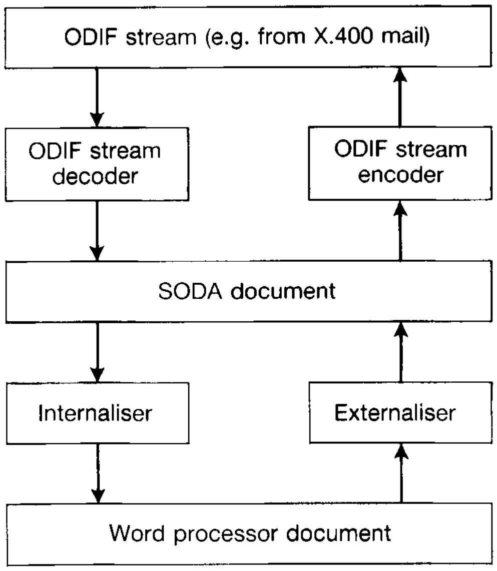


Fig. 5 Diagram of Conversion Architecture

correct representations. Since SODA is a file format, new code modules for externalisation and internalisation can be extensively loop tested via SODA before starting relatively more expensive and less convenient validation through external agencies.

12.1.2 ODIF conversion The ODIF analyser and generator are based directly on the published grammar of ODIF, using an ASN.1 parser. This parser is part of a generally available suite of OSI construction tools called ISODE which is distributed by a number of agencies.

The ISODE tools permit "C" code to be interpolated in the grammar where required to handle the SODA objects which correspond to the grammatical elements. Thus the close association with the ODA specification is maintained.

The ISODE tools also permit a "pretty-printer" to be constructed which parses ODIF, showing all the grammatical elements in printable form. Where syntax errors are found, the context of the error can readily be seen.

12.2 SODA implementation details

12.2.1 Using the interface The software components developed by the project according to the architecture are: company specific proprietary format conversion software; a portable ODA Storage Manager supporting a data structure interface "SODA" (providing random access to storage structures representing ODA constituents); portable software for converting ODIF to SODA and vice versa.

12.2.1.1 Constituents As explained in the section on the structure of an ODA document, such a document is made up of data elements called constituents. Each constituent may have one or more of a repertoire of attributes, some of which relate that constituent to others in the document. These relationships express the structure of the document. Thus the hierarchy of chapters containing sections which contain paragraphs which may contain footnotes will be reflected in constituents which identify their immediately subordinate constituents.

In SODA, each variety of constituent is represented by a "C" structure which contains fields which represent the attributes which are permissible for that constituent. Compact attributes are contained in these fields while longer and complex attributes are found by an indirection mechanism.

The following ODA constituents are supported by the corresponding data structures named below:

Constituent	*Equivalent SODA structure*
Document profile	Profile
Layout object class	LayoutClass
Layout object	LayoutObject
Content portion	ContentPortion
Logical object class	LogicalClass
Logical object	LogicalObject
Presentation style	PresentationStyle
Layout style	LayoutStyle

12.2.1.2 Random access A prime requirement for the SODA interface was that it be efficient, that is, the programs using it could be written to run at reasonable speed with cheap processors in limited memory. To minimise the movement of data in memory, reference to soda objects by the ODASM code and by user code refers to the same areas of memory. To make calls across the interface, only pointers have to be passed (and some parameters), not the bulk of the attributes.

Allowance is made for the fact that a large document may occupy too much storage for all of its parts to be accessible simultaneously. So that this is not a problem even on computers which have no virtual store mechanism, SODA includes features to control the amount of store that may be occupied by accessible constituents. This is arranged by the use of an open and close protocol for relinquishing and reusing the store that each SODA object occupies.

The visibility of the internal structures across the SODA interface offers opportunities and responsibilities. The former is principally that additional fields can be added to the SODA structures, in order to hold intermediate processing results. This has the consequence that the ODASM code must be recompiled, but this is the only cost of such simple extensions. Conversely, the latter implication is that user programs must not disturb the housekeeping data that is in the structures and is not part of the interface.

12.2.2 Externalisation (word processor to SODA) During externalisation the converter code examines the word processor's internal representation of a document and uses the SODA interface to assemble a file store representation of the equivalent ODA document. The ODIF bytestring generator can subsequently be used to produce the corresponding ODIF document.

(a) Initialise the document file by calling the function ‘createDocument’.
(b) Call functions to create “C” language data structures representing each of the required ODA constituent descriptions.
(c) Set values in data structure members corresponding to ODA attributes. Note that appropriate ODA identifiers must be created only to identify styles and classes – they are not needed for logical objects because the ODIF bytestring generator can synthesise them.

(d) Close the constituents as soon as they are completely defined – this allows them to be copied to disc and store to be released.
(e) Close the document file by calling the function 'closeDocument' – this ensures that all constituents are secured to filestore.

12.2.3 Internalisation (SODA to word processor) During internalisation the word processor code uses SODA to access an existing document and builds an equivalent document in its own internal representation. The SODA document may be the file store representation resulting from a received ODIF document which has been internalised by the ODIF bytestring analyser.

(a) Initialise access to the SODA document file by calling the function 'openDocument'.
(b) The profile description is opened by use of the function 'openProfile'. This constituent must be opened before any other constituent. In addition, since access is usually required to the Profile throughout processing of a given SODA file, it is generally better to open the Profile at the beginning of processing and to leave it open throughout, only closing it just before the call to 'closeDocument'.
(c) Constituent descriptions (any existing object, class, style, or content portion) can be opened in any required order by using the 'openDescription' function. This takes an OBJID value to identify the constituent description and it returns a pointer to the appropriate data structure.
(d) Read ODA attributes values from the corresponding members of the constituent data structures.
(e) Close the constituents – this allows their store to be reused.
(f) Close the document file by calling the function 'closeDocument'.

13 Summary

This article has described the demonstrations of electronic document interchange in order to place the international standard "ODA" in context. An overview of Office Document Architecture was presented to show how documents are constructed.

The need for a random-access form of stored documents was explained, and also how this leads to a requirement for an Application Program Interface to manipulate these documents.

It explored the implications of having a published interface for creating and manipulating ODA documents, with toolkits supporting them. A consequence is that the interface does not imply a particular format for stored documents, but that the toolkit does.

Finally, the article presented the ICL solution "SODA", and the ODA toolkit that it supports. Current and potential uses of the toolkit were described. The approach has been highly successful and will continue to the basis for work within the current project and, it is hoped, subsequent ones.

14 Glossary

API "Application Program Interface":
A set of functions that can be called from a computer program; in the context of this paper in order to create, maintain or use an ODA document.

Conversion:
The process of transforming information from one computer representation ("format") to another.

DAP:
Document Application Profile (see section 5).

Defensiveness:
The employment of thorough checking of input parameters to ensure that results are as expected and are not inexplicable.

Language binding:
The specification of the complete syntax to be used to represent each function of a programming interface (API) in a chosen computer language.

15 Bibliography

"Introducing ODA", Ian Campbell-Grant, ICL Technical Journal 5(4) November 1987, pages 729–742.
"X.400 – International information distribution", Dorothy M. Elliott, ICL Technical Journal 5(4) November 1987, pages 754–760.
ISO 8613, "Office Document Architecture (ODA) and Interchange Format".
ECMA 101, "Office Document Architecture (ODA) and Interchange Format" (copies of this version are available free from ECMA).
CCITT Recommendations T.410 series, "Open Document Architecture (ODA) and Interchange Format".

Human-Human Co-operation and the Design of Co-operative Machines

Michael Smyth
Anthony Clarke
LUTCHI Research Centre, Department of Computer Studies, Loughborough University of Technology

Abstract

The paper contrasts co-operation on mental tasks, as practised by groups of people (with or without computer assistance) with that which might be practised by a single individual interacting with a computer. The known benefits of interpersonal cooperation within a group are reviewed from the literature. A set of mechanisms is proposed whereby a single person may cooperate with a computer. These mechanisms have been studied experimentally at Loughborough University of Technology under Alvey research project number MMI/062 that also involved ICL. A set of tools were constructed that allowed the proposed mechanisms to be exemplified in practice in several ways, the simplest of which is briefly described here; it concerns the layout of furniture in an office under stated constraints. Some general conclusions are drawn on the potential benefits (and possible limitations) of catering for human/computer co-operation using the proposed mechanisms.

1 Introduction

As information technology (IT) techniques grow in sophistication, software designers are attempting to build solutions targeted toward increasingly complex tasks. The result has been the evolution of a novel user category, the highly skilled, but not necessarily computer literate, professional user. Professional users may be characterised as having detailed, task specific knowledge and the requirement/desire to undertake a wide variety of complex tasks. Human-Computer Interaction (HCI) practitioners are currently investigating novel machine interaction methods and paradigms which enable users to perform the complex activities required by the task, whilst keeping time taken to learn the system to a minimum. Set against this background there is a growing awareness among the IT community that successful system design will accord the interface between the user and the application software an increasingly pivotal role. This paper describes initial attempts to identify and apply the concepts of human-human co-operation as a technique for improving interaction with machines.

1.1 A question of software usability

Software design is no exception to the general theme that design is concerned primarily with the production of artifacts. In this context an artifact is a product or tool to assist a user in the performance of a specific task or function. In order to develop successful software the designer must first understand the task at which the application is targeted. Such understanding involves the identification of terms and terminology specific to the task and how these are used and combined during the task. A second and equally difficult problem faces the system designer, how to design a system which caters for the variation in user expertise and expectation both at the outset and during the use of computer based tools. It is the very generality of computing and its potential application domains which places such a burden on the system designer, whose ultimate responsibility must be to the people who will use the eventual solution.

Software usability will be determined by the user during interaction with the machine. For example, whether a task can be performed using terms and procedures familiar to the user, whether the system is easy to learn, whether occasional users can quickly recall how the system operates and whether the system provides adequate feedback and error protection coupled with a consistency of presentation throughout the interaction. In short, does the software solution enable the user to increase task performance. The key to the design of usable software is an understanding and empathy with the user group and the task they perform. The task requirement and characteristics of the user group should be central in the software design cycle. The importance of this position has led to the phrase User Centred System Design (Norman and Draper, 1986). Cognitive psychologists aim to provide software designers with tools and techniques for achieving usability.

The growth of cognitive psychology has paralleled that of computer science. Cognitive psychologists were quick to realise the potential of computing as a metaphor for human information processing. Indeed the current models of human memory and attention have been particularly influenced by this association. The result has been the development of a number of predictive models of human behaviour. Coupled with design guidelines (e.g. Smith and Mosier, 1984 and Shneiderman, 1987) and the advocacy of a more central role for the end user, these models have allowed software designers to incorporate a greater sensitivity to human behaviour in their solutions.

Several of the predictive models spawned by cognitive psychology have focussed on how people perform tasks. For example, the Human Information Processing Model (Card, Moran and Newell, 1983) and the Keystroke Model (Card, Moran and Newell, 1980), attempt to draw conclusions about task performance (i.e. how long a task will take and what are the processing requirements). While Task Action Grammar (Payne and Green, 1986) and Command Language Grammar (Moran, 1981), attempt to identify the tasks to be carried out and, given the available command sequences, how these might be achieved. Finally, models such as GOMS (Card, Moran and Newell, 1983) and PUM, referred to by Simon (1988), analyse expert behaviour, albeit in routine tasks. In general these models apply quantifiable characteristics of how humans process information in artificially constrained tasks and attempt to inform the software designer how best to structure communication between the user and the system.

Other models developed in cognitive psychology have studied man-machine interaction and are generally referred to as layer models. These models attempt to represent how a high level idea on the part of the user is broken down into composite parts and, at the lowest level, communicated to the machine where it is recombined into actions within the machine to a level of detail that can be recognised by the user. Examples of such layered models include those developed by Clarke (Clarke, 1986), Moran (Moran, 1981), Nielsen (Nielsen, 1984) and Foley and van Dam (Foley and van Dam, 1982).

As a result of technological advances and the specific requirements of particular application domains five major styles of interaction have evolved. In no particular order these are form fill, natural language, direct manipulation, menu driven and command language. What is perhaps most notable is that rarely, if ever, have interaction styles been developed as a direct outcome of attempts to model cognitive processes in human behaviour. The models outlined above may assist with design decisions, but cognitive psychology does not drive the development of interaction styles. A possible reason for this might be the associated difficulty encountered when attempting to apply models of cognition during the software design cycle. Although laudable in their objectives the sheer complexity of the models of human behaviour produced so far, coupled with the amount of development time associated with their use, have limited their direct application in software design. Their input appears to be in a supportive role, that is, the results of cognitive modelling are used as supportive evidence for the development of novel, mainly technology driven, interaction techniques, rather than directly influencing interaction per se. With the exception of natural language interaction, it would appear that while communicating with machines, an implicit emphasis is placed on the user to adapt to the interaction style selected by the system designer. It is true that human interaction is much more complex and rich and it is therefore within the user's capability to adapt to the machine's poverty of communication, but if this is to occur in a positive sense, it is essential that the user develops a correct mental model of how the system behaves. The construction of such a model is made all the more difficult by the use of interaction styles which are different from those of human-human interaction with which the user is familiar. As observed by Gaines (1981), 'to make the system understandable is to maximise the possibility of the user forming with the minimum of effort a model of the system which aids his effective use of it'. Such models are difficult to achieve if the interaction style of the system bears no resemblance to that of the human user. To some extent this problem has been alleviated by the use of metaphors during interaction, which act as a link to the user's previous experience thereby helping in the prediction of machine behaviour.

An alternative approach is to attempt a closer parallel to human interaction within the artificial constraints imposed during interaction with machines.

This idea is not new, indeed Scrinivasan and Dascher (1977) stated that, 'we need to develop a language structure that will embrace a broad diversity of the kinds of communication that users normally use'. What is not obvious is how such aims might be translated into usable procedures to assist software designers. A possible route, explored in this work, might be to use social psychology to provide a clearer picture of how humans communicate during complex tasks. The work in social psychology is not seen as a solution in itself, but as another perspective from which to view the problem of how to design more usable computer systems.

1.2 Social psychology and the study of interpersonal behaviour

Interpersonal behaviour has been extensively studied by social psychologists. Primarily it is concerned with the identification of factors which both motivate behaviour and affect and modify communication based on the individual and mutual needs and requirements of participants in a task. Such work could provide a rich source of data for the design of computer systems which aim to take advantage of existing interaction styles among humans. Human communication is a highly complex form of interaction, motivated by both task-related and social goals (Murray and Bevan, 1984). Tasks requiring either joint or dependent actions from participants can lead either to co-operative or to competitive behaviour. Several factors have been identified which affect this choice of behaviour strategy; these include the nature of the task, the expectations of the participants and the personal goals of either party. Such is the distinction that it has prompted some researchers to claim that there exist two basic personality types in humans, those expecting co-operation from others and those expecting competition (Kelley and Stahelski, 1970). If co-operation is to be the model of machine interaction, and not competition, it is essential to identify the task-related, and the social factors which induce such behaviour among humans.

1.2.1 Factors which induce and maintain human co-operation Central to the maintenance of co-operative behaviour between humans is the existence of superordinate goals. These are goals which are compelling for the individuals involved, but that cannot be achieved by one individual by his or her own efforts. Such work was initially reported by Sherif and Sherif (1953), and later replicated by Blake and Mouton (1962), and by Blake, Shepard and Mouton (1964), for a variety of tasks. The existence of superordinate goals would appear to be one of the strongest factors in the development and maintenance of co-operation, both between individuals and groups.

Co-operation is not a fixed pattern of behaviour, but is a changing, adaptive process directed to future results. The representation (and understanding) of intent by every participant is therefore essential to co-operation, and so the role of communication in co-operation is important, (Oberquelle 1983, 1984). Grice's 'co-operative principle' may be considered here, and his 'rules' of Quality, Quantity, Relevance, and Manner can serve as guidelines to the requirements of communication in co-operation, (Grice 1975). Style, as well as content, of communication between parties facilitates co-operation, but does not create co-operation, (Shure et al., 1965). Communication which can be identified by the other party as reducing potential threat is best suited to increasing co-operative behaviour, (Deutsch and Krauss 1960). A high level of communication is implicit in co-operative behaviour and it may be defined as 'a complex of social actions for the purpose of mutual understanding and for allowing co-ordinated actions to occur'.

In reality co-operative behaviour between humans is maintained by a combination of such factors as open communication, exchange of information and the existence of superordinate goals.

1.2.2 Task factors which support co-operation It may not be immediately apparent that not all tasks allow the potential of co-operative behaviour to be realised. For instance, tasks which require a single correct answer (e.g. mathematical problems) tend to provide little scope for co-operation. Whereas situations where more than one alternative is sought, provide a problem solving environment which favours co-operative behaviour. Empirical evidence to support this position has been provided by two experiments. Firstly, Thorndike (1939), conducted an experiment to test the hypothesis that the superiority of the group over the individual will be greatest for tasks which afford a wide range of possible solutions. In general the hypothesis was confirmed. While Husband (1940), found that pairs were superior to individuals when working on problems requiring some originality or insight, but not on more routine arithmetic problems.

1.2.3 Advantages of co-operation during problem solving Empirical studies of co-operative behaviour between humans appear to support the contention that joint effort during complex problem solving tasks is reflected in both the process of achievement and the quality of solution (Deutsch, 1949, 1968, Laughlin et al., 1969). While studying group performance, of which co-operative behaviour is an example, Davis (1969) commented that if each person possesses unique but relevant information, and the task requires several pieces of information, then pooling of this information will allow groups potentially to solve problems that an individual cannot attack successfully. He further stated that if the emphasis is on achieving a correct, good or early answer, then a group has a higher probability of achieving this aim (other things being equal) than an individual. Dynamic social interaction during complex problem solving may reinforce or negate existing beliefs and support the formation of new attitudes between and in the participants. It is this potential for mutual growth that makes co-operative behaviour so important in human-human problem solving.

1.3 Human computer co-operation

Co-operation is viewed as an active process. A machine employing this technique must participate overtly with the user during a task. A co-operative machine, while remaining focussed toward an expressible goal, will prompt the human partner to adopt a more divergent style of thinking during problem solving. It is suggested such interaction will foster a greater interdependency between the human and the machine and, as a consequence, increase the quality of solution and encourage greater user satisfaction.

A principal element of co-operative behaviour during problem solving is the creation of an environment, where solutions can be refined by logical argument and the resolution of differing perspectives. Through these discussions the very essence of the problem is revealed. Examples of techniques developed to foster divergent thinking between humans include Brainstorming (Osborn, 1953) and Synetics (Gordon, 1961). Essential to this view of the co-operative dyad is the ability of either party to generate and communicate alternative solutions, as it is these which spark the iterative process of solution implicit in co-operation (Broadbent, 1973). In short, different but sympathetic beliefs are vital to a successful and productive co-operative relationship. The importance of such a dynamic interaction during complex problem solving behaviour is reflected in Feyerabend's (Feyerabend, 1965) statement that 'progress can only be brought about by the active iteration of different theories'. Although recognised as only one facet of the complex relationships involved in human-human co-operation, the generation of alternative solutions is felt to provide a starting point for representing the co-operative relationship between a human and a machine.

The focus of the work outlined here is to develop a single-user co-operative mechanism where the generation of a satisfactory solution could be enhanced by a machine having the ability to generate alternative and supplementary information based on a solution proposed by the user. Any attempt to harness the potential of human-human co-operation and apply it to HCI will result in viewing the machine as an active agent and not purely as a provider of information on request.

Such machine generated alternatives, it is contended, could act as catalysts and so play a more active rôle in the formation of ideas by changing the context in which the user perceives the problem, thereby providing what Jones (1970), called a 'greater perceptual span'. The aim of the work is to provide a problem solving environment aimed at professional users based on a co-operative paradigm.

2 The underlying mechanisms of a co-operative machine

Empirical and observational studies of interpersonal behaviour in social psychology indicated a number of factors which both induce and maintain co-operative behaviour between human partners during problem solving tasks. This raised the question of how best to translate what are essentially behavioural characteristics into mechanisms which could successfully be represented in a machine.

After prolonged debate, three factors were selected that the group thought reflected the underlying processes of co-operative behaviour. These were,

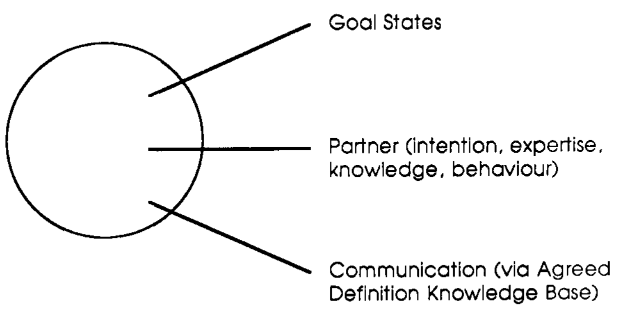

Goal States
Partner (intention, expertise, knowledge, behaviour)
Communication (via Agreed Definition Knowledge Base)

Fig. 1 Underlying Mechanisms of the Co-operative Machine

firstly, the existence of superordinate goals, referred to as Goal Oriented Working, secondly, a model which contained knowledge about the specific task domain and would thereby represent the knowledge of the computer partner, referred to as a Partner Model, and, finally, a language common to both the co-operating parties, referred to as the Agreed Definition Knowledge Base. Several factors associated with co-operative behaviour, for example the role of social goals as a reward mechanism for undertaking co-operation, were considered specifically human traits and were discounted as critical to human computer co-operation.

Although acknowledged as representing only a subset of the complexity of behaviour displayed during human-human co-operation, it was felt that a machine representation of goal oriented working, a partner model and an agreed definition knowledge base would provide an initial architecture from which to study human computer co-operation.

2.1 Goal oriented working

If a co-operative dyad is to succeed, be it either human-human or human computer, there must be agreement, formally stated at the outset of the interaction, on the goal to be accomplished. Two parties working toward different or conflicting goals will never achieve co-operation. It is obviously essential for co-operation that the parties, firstly, know what the desired goal is and, secondly, work towards attaining it.

A goal may be defined as the intended state of an object or the intended relationship between two objects or more. The action of achieving the goal is directed at goal objects, which may either be physical or virtual, and can result in their creation, elimination or modification. Each goal object has a number of attributes, such as colour or dimension, which may be altered in the act of achieving a goal. For example, the goal of sharpening a pencil can be restated in the form of, the goal is to have pencil in a sharpened state. In this case the goal object is the pencil and the achievement of the goal will result in the attributes of sharpness and length being changed. Figure 2 represents diagrammatically the autonomous agents of human user and computer communicating and working toward achieving a goal.

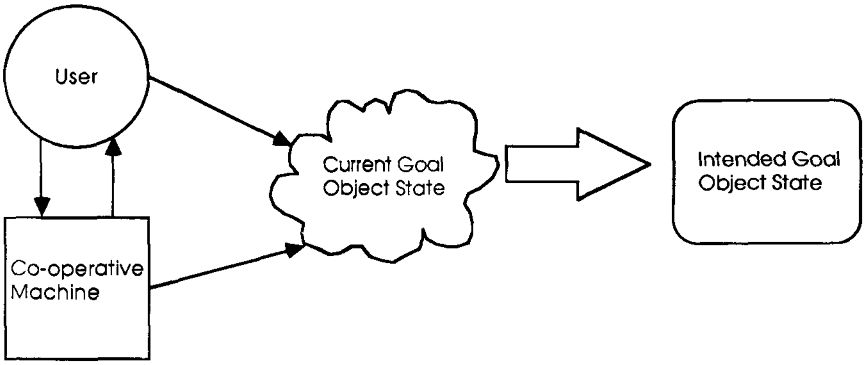


Fig. 2 Representation of Goal Oriented Working

2.2 The partner model

The ability to generate alternative solutions to a problem solving task is an important strand in the complex processes exhibited during co-operative behaviour. In order to represent an equivalent mechanism it is necessary to construct a model, consisting of domain specific rules, which has access to a knowledge base common to both the machine and user. The term adopted for the knowledge base is the Agreed Definition Knowledge Base (ADKB). By applying the model's rules to the ADKB and communicating the result to the user it is hoped to reflect the process of generating alternative solutions seen in human-human co-operation.

In essence, the co-operative machine requires a model, here referred to as the partner model, both to generate alternatives and to facilitate interaction with the user. The partner model is essentially different from an embedded user model, in that it possesses similar domain information to the user, but does not necessarily reflect the working style of the particular end user. The facility to query the rule base of the partner model is essential if the user is to comprehend the processes leading to the alternative solution. It is only through such understanding that the user might begin to incorporate the partner model's techniques in his or her own design solution strategy. Communication of alternative configurations of a design solution in an interactive environment might facilitate the development of a user's task expertise; that is during interaction the human partner could begin to adopt the task skills represented in the partner model. An embedded user model attempts to extract concepts exhibited by the user during the interaction and ultimately mimic or predict the user's solutions, a process felt to run contrary to the philosophy of human computer co-operation. The requirement for a partner model process to be autonomous from the user reflects the essential difference between entities that makes co-operation such a powerful technique for problem solving.

2.3 The agreed definition knowledge base

Central to co-operative behaviour is the ability of either party to communi-

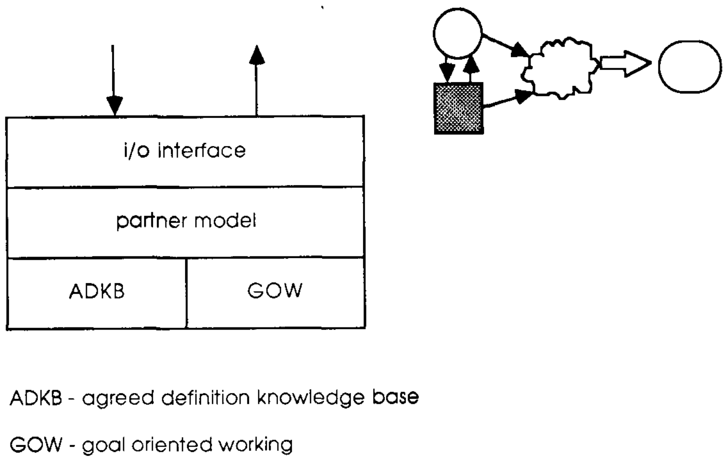

i/o interface
partner model
ADKB
GOW
ADKB - agreed definition knowledge base
GOW - goal oriented working

Fig. 3 Schematic Representation of the Co-operative Machine

cate the current state of the interaction. Thus it is important that both co-operating parties share the same object definitions. This factor is emphasised during human-human co-operation where much time is taken up by the mutual identification and agreement of terms of reference. Definitions do not have to remain static as long as changes are agreed by both parties. If a co-operative machine is to be fully developed it must provide a sufficient breadth of communication to enable the interactive updating of the shared knowledge base.

Figure 3 provides a diagrammatic representation of the three mechanisms which will form the basis of the COM exemplar.

3 A co-operative task metaphor

At the outset of the project there was consensus among the group members that the goal should be to produce general purpose co-operative mechanisms. The idea was that, in principle, co-operative techniques and their associated advantages could be applied to any task. In principle this idea posed little or no real problems during the early discussions as to what did and did not constitute co-operative behaviour. Two potential task domains were included in the original project proposal. These were highway design and electronic circuit board design. Although useful as a focus through which to express the group's early, ill formed ideas, it soon became clear that the overhead associated with representation of knowledge in these domains was beyond the timescale of the project, if they were to support rudimentary co-operation. In fact an investigation of highway engineers (Clarke et al., 1986) revealed that knowledge of the position was not held by a single person but by a group of individuals each with specialist knowledge.

There was a general feeling that design tasks offered a potential application for co-operative techniques. As a result several papers were produced (Shuttleworth, 1988; Smyth, 1987, 1988) which investigated how designers go about generating solutions. In general, it was found that designers are solution focussed, that is, they pose tentative solutions to a problem and by so doing learn about the problem. Through this iterative process of analysis, synthesis and evaluation, the design converges on a satisfactory solution.

As the project progressed it became clear that the early aims of producing general co-operative techniques were impractical. There was a pressing need for an easily understood task familiar to the group, which could act as a vehicle for detailed discussion of co-operative behaviour. As a preliminary to the identification of such a task domain, a series of requirements were identified.

- the task should be recognised as 'real world'
- there should be a finite variety of possible solutions, so as to enable the production of alternatives between the co-operative parties
- task goals should be identifiable and expressible at the outset
- explicit rules should be involved in arriving at solutions
- the task should support the generation of partial solutions
- the task should support graphic based interaction

The problem domain selected was room layout design, which is one instance of spatial design problems.

The importance of this decision should not be understated as it provided an achievable focus for discussion and the eventual implementation of a co-operative machine exemplar. It is stressed here that the eventual software exemplar was never seen as a practical room layout application, as many more sophisticated solutions already existed. It aimed merely to use the application as an expression of the three underlying mechanisms of co-operative behaviour; room design acted as a task metaphor for co-operation between humans and computers.

3.1 The design process – a vehicle for co-operative problem solving

It is commonly found that high level design goals are expressed in terms of constraints and requirements; they may be a result of the medium in which designers choose to work, or be caused by client or user requirements, or finally they may result from the task itself. Indeed design can be described as the successive application of constraints until a unique product is left. Laurel (1986), states that, rather than restricting the process of design, 'constraints provide the security net that enables people to make imaginative leaps'. The potential advantages of co-operative behaviour in the design process have been shown by several researchers. Middleton (1967) observed that testing hypotheses benefited from group activity; he stated, 'the distinction between original thought, which is a lonely activity, and the testing of hypotheses, is a logical process where the group can participate with advantage'. Thus it would appear that design is ideally suited to benefit from co-operative behaviour.

The next question was, are there general rules of design and in particular spatial layout, which could be incorporated within the partner model? Such rules would form the mechanism whereby the co-operative machine could generate meaningful alternative solutions.

Designers are often confronted with the task of choosing, from among seemingly limitless possible arrangements, an aesthetically satisfying composition of objects in a given space. Some choose to rely on the intuitive application of their skills. Tufte (1983) points out, in reference to graphic design, 'there are no compositional principles on how to create that one graphic in a million'. Nor are there any in other fields of design. Nonetheless, some designers cut down the number of possibilities in compositional tasks by applying rules to govern the placement of objects in a few initial sketches. They can then choose to enforce the rules more or less strictly, apply further rules and develop compound effects, in response to the resulting compositions. One fundamental rule of composition is to arrange objects along lines of symmetry; another, that has been applied for centuries, is that of the Golden Section (Holt, 1971). Rules of this nature were agreed as essential to the partner model if it were to co-operate successfully with the designer. Since the role of the exemplar was paradigmatic and called for only a few archetypal rules to illustrate the principle, it was not considered appropriate to research the area exhaustively.

With the adoption of the room layout as a metaphor, it was considered that the requirements of the agreed definition knowledge base could be met by including a subset of generic furniture objects to be acted on by both goal definitions and rules within the partner model.

4 An exemplar of human/computer co-operation – room layout

At the outset of the implementation of this co-operative exemplar, two system design decisions were made,

- when possible to, put control of the interaction in the hands of the user.

This refers specifically to stages during the interaction requiring judgements, for example, deciding what configuration of objects constituted a satisfactory layout.

- to minimise the requirement for explicit broad band communication between the user and the machine, and to instead adopt an implicit, graphically based, method of communication.

The exemplar was implemented in C-Prolog mounted on a Hewlett Packard workstation and used a general purpose environment built by the Architecture Stream of the Human Computer Co-operation project.

4.1 The interface

The philosophy adopted by the group for the exemplar interface emphasised visual representation of task information. Observation of two human designers co-operating during a complex problem solving task indicated a requirement for two dedicated graphical windows, which represented a view of the floorplan of both the user and the partner respectively. This simple decision overcame the problem of presenting alternative solutions to the user, while not causing a distraction from the current task. Other user actions, such as the construction of goals, the prioritisation of the partner model rule base and the creation of objects were catered for by selection of appropriate software buttons.

The exemplar has two modes of operation, solve and interactive. In the solve mode the partner model will immediately generate an alternative solution based on the current active goals and the location of objects in the user's solution. The solve mode was primarily used during software development and was never considered to be central to the development of the co-operative relationship. Whereas in the interactive mode the partner model is controlled by the user's choice of object locations, a method thought to characterise, albeit at a simplistic level, the equivalent relationship in human-human co-operation.

4.2 The agreed definition knowledge base

The room design metaphor contained a deliberately constrained, hand crafted, object set which was presented visually to the user throughout the interaction. Selection of an object type caused the creation of an instance of that object to be placed in the inventory. The user was then at liberty to incorporate that object instance in a goal construct or simply place a graphic of the object in the user window at the desired location. Both the user and partner windows represent a notional floorplan. As objects are located in the user window the partner model generates alternative positions based on its rules and displays its option in the partner window. Throughout this process parallel agreed definition knowledge bases are continually updated as the user and partner generate alternatives of the available objects. The standard object data structure is as follows,

```
object ( < user/partner>,
         object-type (object-instance),
         location,
         <permanent/moveable>).
```

Transfer of data from the databases was at the discretion of the user, who could either freeze objects in the user's window, thereby disabling the effect of the partner model, or transfer object configurations generated by the partner model to be incorporated in the user solution. It was felt that both these techniques resemble those observed in human-human co-operative behaviour.

4.3 Goal oriented working

A software technique developed within the Architecture Stream of the HCC project, which was used extensively throughout the project, provided the enabling technology for the representation of goal oriented working in the exemplar. It enabled the user to construct spatial relationships between objects which were then translated into the Prolog rule base which enabled the partner model to manipulate them. The provision of a method whereby the user was able to construct goals visually was central to the development of the software exemplar.

A representative number of spatial relationships were hand crafted in the system, and users were able to construct and delete goals as required during the interaction. The relationships were object sensitive, for example the concept of nearness is quantitatively different when applied to a desk/wall combination than to a desk/chair combination.

The goal oriented working mechanism also included simple error checking which alerted the user to possible sources of goal conflict. Once the goal list has been created the software calculates the order in which to place the objects. If, for example, the following goals were created,

phone	on	desk
desk	near	wall

the desk would have to be positioned first before the initial goal could be satisfied. If goal fulfilment was deemed to be object independent the order of creation was the deciding factor. Examples of user constructed goals are as follows,

cabinet1	near	wall2
bookcase1	near	wall3
bookcase1	near	wall4
desk1	near	window1
chair1	in-front-of	desk1
chair2	in-front-of	desk2

4.4 The partner model

The partner model consists of a subset of design knowledge concerned with the proportionality of objects within a finite space, in this case the location of furniture within a room. Examples of the rules in the partner model include, symmetry, golden section and safety, the latter being chosen as it reflected a different style of rule. The safety rule caused an access route from the window to the door always to be left clear. At each stage in the user's development of a solution the partner model can interrogate the resulting object configuration and apply its rules thereby generating, within a number of user defined goals, an alternative composition of the objects based on the user's solution.

The design solution generated by the partner model is neither right nor wrong, but is a viable, alternative configuration of the user's objects developed using a set of rules different from that of the user and its aim is to act as a spur to the user's imagination. Incorporated in the partner model's solution strategy is a mechanism for identifying and rejecting alternate solutions which break fundamental constraints implicit in the task domain. For example, objects must remain within the bounds of the room and must not occupy the same space.

The user could alter the order in which the partner model rules were applied. This enabled the user to view a number of alternative object configurations each generated by a partner model with a different emphasis or style. Division of labour was thus placed in the hands of the user so, for instance, the user could indicate that he wished the partner model to place a high priority on safety, so leaving himself free to pay more attention to the stylistic details of the solution.

4.5 Role of the software in the development cycle

The development of the exemplar was of great benefit to the group's work as it provided a tangible demonstration which led to many fruitful discussions and ideas which could later be incorporated in the software. Toward the end of the project two final year design students performed an informal evaluation and their comments were both valuable and enlightening.

Although at a rudimentary stage in development, the experiences gained during the process only served to confirm the group belief in the potential of co-operative computing.

5 Conclusions

The primary objectives of the work described in this paper were,

- the identification of the mechanisms central to the design of co-operative machines
- the exemplification of these mechanisms in software

5.1 Conclusions from software development

The degree of co-operation achieved by the machine will be determined by interaction between the underlying mechanisms, not by their individual actions. How this resulting machine behaviour will be manifest to the user will be dependent on the interface. In the context of the room design metaphor a number of interface requirements were identified. Adopting the philosophy of visualisation of knowledge, the interface should enable the user firstly, to define task objects and secondly, to support the interactive definition of goal relationships. In short, the interface of a co-operative machine should enable the user to enhance the agreed definition knowledge base and, thereby, more directly focus the process of generating alternative solutions and so increase the degree of co-operation perceived by the user.

- end user manipulation of the data shared with the partner is a technique central to the development of co-operative machines.

5.1.1 The partner model Adopting the technique of modelling a co-operative partner in the machine allowed an important conclusion to be drawn, namely,

- to be able to produce alternative solutions, the partner model rule base must remain autonomous from the user if the co-operative machine is to avoid the pitfall of simply mimicking solutions generated by that user.

Adherence to the above point raises some interesting issues about the nature of modelling partners in single user co-operative machines. Co-operation between humans is a dynamic relationship, as it is assumed that both parties are learning/reinforcing the concepts being expressed by their partner during the interaction. Over a sufficient time period it is foreseeable that the co-operative relationship could become static, as, without the introduction of new sources of information, a commonality of views might arise between partners so reducing the effectiveness and, ultimately the creative potential of the relationship. In human-human co-operation additional information can come from a multitude of sources as reflected, for example, in the background, education and personal preferences of the individuals. On the other hand the current machine model of the co-operative partner is static. A single-user system does not provide an alternative source of information necessary for the partner model to develop during the interaction, and the use of information gleaned from the user during interaction, as in dynamic, embedded-user models, would reduce the difference between partners and remove a plank vital to the co-operative relationship. In short, the ability of the human partner to learn during a co-operative interaction ensures a limited period of usefulness for a static partner model in a single-user system. As the user becomes aware of the partner model rule set, the machine generated alternatives become predictable and no longer a spur to the user's imagination.

In the context of human computer co-operation a partner model is proposed as a technique to represent the essential difference between the user and the machine necessary to realise some of the dynamic qualities of co-opertive problem solving.

5.2 General conclusions

The work undertaken in this project has highlighted several important issues felt to be central to the development of future co-operative machines.

- the depth of knowledge required to support co-operative working suggests that the technique is domain specific
- what constitutes co-operation varies between tasks. The existing problem solving process adopted by human partners must be clearly identified before attempting to build a co-operative machine
- the mechanisms of Goal Oriented Working, the Partner Model and an Agreed Definition Knowledge Base, as described in this paper, provide an outline architecture for co-operative machines
- it is the interaction of these software mechanisms and how the resulting behaviour is manifested to the user, via the interface, that will determine the extent of machine co-operation perceived by the user.

The issues raised during this project are central to the development of computer systems which aim to actively participate with the user during complex problem solving and as such provide a platform for future research work.

6 Acknowledgements

The work described was funded by the Alvey programme MMI, Human Computer Co-operation (MMI/062). Thanks are due to our colleagues at the LUTCHI Research Centre, in particular, Steve Bell, Mark Buckley, Tom Bayley, Andrée Woodcock with Mark Shuttleworth from ICL.

7 References

BLAKE, R.R. and MOUTON, J.S. (1962) The inter group dynamics of win-lose conflict and problem solving collaboration in union management relations, in M. Sherif (ed.), Intergroup Relations and Leadership, Wiley, NY.

BLAKE, R.R., SHEPARD, H.A. and MOUTON, J.S. (1964) Managing inter group conflict in industry. Houston: Gulf.

BROADBENT, G. (1973) Design in Architecture, John Wiley & Sons, London.

CARD, S.K., MORAN, T.P. and NEWELL, A. (1983) The Psychology of Human Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, NJ.

CARD, S.K., MORAN, T.P. and NEWELL, A. (1980) The Keystroke-Level Model for User Performance Time with Interactive Systems, Communications of the ACM, Vol 23 No 7, 396–410.

CLARKE, A.A., WOODCOCK, A. and McDAID, E. (1986) The Highway Engineer; a Typical User? HCC Internal Report HCC/L/9, Loughborough University of Technology, UK.

CLARKE, A.A. (1986) A three-level human-computer interface model, Int Jour Man-Machine Studies, 24, 503–17.

DAVIS, J.H. (1969) Group Performance, Addison-Wesley.

DEUTSCH, M. (1949) A Theory of Co-operation and Competition, Human Relations, 2, 129–152.

DEUTSCH, M. (1949) An experimental study of the effects of co-operation and competition upon group processes, Human Relations, 2, 199–231.

DEUTSCH, M. (1968) The effects of co-operation and competition upon group processes. In D. Cartwright and A. Zander (eds), Group Dynamics, Harper and Row, New York.

DEUTSCH, M. and KRAUSS, R.M. (1960) The effect of threat on interpersonal bargaining, Jour of Abnormal and Social Psychology, 61, 181–9.

FEYERABEND, P. (1965) Consolations for the Specialist. In I. Lakatos and A. Musgrave (eds), Criticism and the Growth of Knowledge, Cambridge University Press.
FOLEY, J.D. and van DAM, A. (1982) Fundamentals of Interactive Computer Graphics, Prentice-Hall, Englewood Cliffs, NJ.
GAINES, B.R. (1981) The technology of interaction – dialogue programming rules, Int Jour Man-Machine Studies, 14, 133.
GORDON, W.J.J. (1961) Synetics, the development of creative capacity, Harper & Row, New York, Evanston, London.
GRICE, H.P. (1975) Logic and Conversation, in Cole, P. & Morgan, J.L. (eds) Syntax and Semantics, Vol 3 'Speech Acts', Academic Press, London.
HOLT, M. (1971) Mathematics in Art, Studio Vista, London.
HUSBAND, R.W. (1940) Co-operation versus solitary problem solution, Jour of Soc Psychology, 11, 405–9.
JONES, J.C. (1970) Design Methods: seeds of human features. John Willey, N.Y.
KELLEY, H.H. and STAHELSKI, A.J. (1970) Social interaction basis of co-operators' and competitors' beliefs about others, Jour of Personality and Social Psychology, 16, 66–91.
LAUGHLIN, P.R., McGLYNN, R.P., ANDERSON, J.A. and JACOBSON, E.S. (1968) Concept attainment by individuals versus co-operative pairs as a function of memory, sex and concept rule, Journal of Personality and Social Psychology, 8, 410–7.
LAUREL, B.K. (1986) Interface as Mimesis, in Norman, D.A. and Draper, S.W. (eds) User Centered System Design: A New Perspective on Human-Computer Interaction, Lawrence Erlbaum Assoc, Hillsdale, NJ.
MIDDLETON, J. (1967) Group Practice in Design, London.
MORAN, T.P. (1981) The Command Language Grammar: A representation for the user interface of interactive computer systems, Int Jour Man-Machine Studies, 15, 3–50.
MURRAY, D. and BEVAN, N. (1984) The Social Psychology of Computer Conversations, Interact'84, 268–73.
NIELSEN, J. (1984) A virtual protocol model for computer-human interaction, DAIMI PB-178, Aarhus University: Computer Science Dept.
NORMAN, D.A. and DRAPER, S.W. (1986) User Centered System Design: A New Perspective on Human-Computer Interaction, Lawrence Erlbaum Assoc, Hillsdale, NJ.
OBERQUELLE, H., KUPKA, I., MAASS, S. (1983) A view of human-computer communication and co-operation, International Journal of Man-Machine Studies, 19, 309–33.
OBERQUELLE, H. (1984) On models and modelling in human-computer co-operation. In G Van der Veer et al (eds) Reading in Cognitive Ergonomics, Springer Verlag.
OSBORN, A.F. (1953) Applied imagination, Charles Scribner's Sons, New York.
PAYNE, S.J. and GREEN, T.R.G. (1986) Task Action Grammars: A Model of the Mental Representation of Task Languages, Human Computer Interaction 2, 93–133.
SHERIF, M. and SHERIF, C.W. (1953) Groups in Harmony and Tension, Harper, NY.
SHNEIDERMAN, B. (1987) Designing the User Interface: Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Co.
SHURE, G.H., MEEKER, L.J. and HANSFORD, E.A. (1965) The effectiveness of pacifist strategies in bargaining games, Jour of Conflict Resolution, 9, 106–117.
SHUTTLEWORTH, M. (1988) Office Layout: A user/task analysis, HCC Internal Report HCC/1/25. Loughborough University of Technology, UK.
SIMON, T. (1988) Analysing the Scope of Cognitive Models in HCI – A trade-off approach, Proceedings of HCI'88, Cambridge University Press.
SMITH, S.L. and MOSIER, J.N. (1984) Design Guidelines for the User Interface for Computer-Based Information Systems, The MITRE Corp, Bedford, Mass.
SMYTH, M. (1987) Toward a Co-operative Design System, HCC Internal Report HCC/L/20, Loughborough University of Technology, UK.
SMYTH, M. (1988) Articulating the Designers Mental Codes, HCC Internal Report HCC/L/24, Loughborough University of Technology, UK.
SRINIVASAN, C.A. and DASCHER, P.E. (1977) Information systems design: user psychology considerations, MSU Business Topics, 25, 51.
THORNDIKE, R.L. (1938) The effects of discussion upon the correctness of group decisions when the factor of majority influence is allowed for, Jour Soc Psychology, 9, 343–62.
TUFTE, E.R. (1983) The Visual Display of Quantitative Information, Graphic Press, Connecticut.

Regulatory Requirements for Security – User Access Control

C. W. Blatchford

Abstract

The Legal profession are divided on what constitutes a computer crime. In particular hacking, the gaining of unauthorised access to a computer is currently not contrary to criminal law in the UK.

Legislation is likely to be drafted for tabling at the next session of parliament.

Whatever the relevance of such legislation in controlling the spread of hacking, successful policing must be paralleled by changes to the way in which information systems are architected and implemented.

This article explores the current architectural problems, both administrative and technical, that need to be solved if computer security legislation is to be meaningful.

This may have a fundamental impact on the way in which Systems planners and engineers view the fitness for use of computer solutions.

Introduction

The Global Telecommunications Village! A hackneyed yet apt description of the society that could be created from the implementation of the various information technology telecommunications and broadcast services (ITT & BB) now being developed.

Anybody who has ever lived in a village, especially in the Celtic fringe of Europe, will realise that village life has its limitations. Paramount is the obstrusive nature of much of human relationships, the lack of personal privacy, the rumours, the half truths, the innuendo's, the witchhunts. Information confidentiality and integrity are at a low premium.

Our Global Telecommunications Village may suffer from similar problems, compounded by the sheer volume and speed of communication. Misinformation could become fact in a few seconds!

There are two major ways of externally enforcing controls on any group of people

- by legislation & regulatory controls
- by reducing the technical opportunity for adverse conditions. (In releasing, falsifying or destroying information.)

It is essential to have a balanced mixture to complement the basic human administration and relationships in society. Many of these controls are directed at the user interfacing with the information system in an authorised and correct manner.

Legislation/Regulation

There is a limited UK regulatory framework in which to place the access control requirements to electronic information systems.

In simple terms however many basic principles about respect for the property and privacy of others are covered in legislation although the offences may be civil not criminal and hence carry very little deterrent value.

There are some possible analogies with "trespass", "theft", breaking and entering and the traditional "property" offences but these are usually unsatisfactory and not well suited to the high technology issues that are being confronted in a computer literate world.

The most difficult situation is with "hackers", people attempting to obtain unauthorised access to computer systems – who apparently do no more than browse the information system without any obvious attempt to cause obstruction, damage etc.

In the UK, like many countries, the legal profession are divided on what can constitute a crime in respect of hacking. The House of Lords decision on Regina v. Gold and Chifreen (1988), in effect, implied that just gaining unauthorised access to a computer is not contrary to criminal law. The Law Commission Working Paper No. 110 – Computer Misuse (1989), generated a flurry of analysis into the legislative problems of unauthorised computer access and use, and the difficulty of defining and implementing controls.

Although there is a general consensus that an all embracing statute on computer crime is not appropriate, there have been calls from organisations such as CBI, ICA to make hacking (for whatever purpose) a crime in the UK.

ICL has been considering this problem for some time and has commented on the Law Commission's Working Paper. It agrees that there should be an offence of hacking, however this should be based on the purpose of protecting the integrity of the computer systems and not related to some intrinsic property of value of the information being processed.

The system must utilise a recognised device or coding system designed to prevent unauthorised use of the computer. This would avoid the drafting of legislation that would be too wide to be effective. The concept of a barrier to access/cross in a computer system is fundamental to good information systems design.

ICL has come to the conclusion that a criminal offence of unlawful and unauthorised access to a computer system should be created, and that such illegal access should be penalised by an appropriate fine and/or imprisonment.

The revised Law Commission report on Computer Misuse embodying various inputs, including those of ICL has been published. It went via the Lord Chancellor to the House of Lords in late 1989, the aim being to approve legislation in the 1989/1990 session of Parliament. H. Colvin MP is to raise a Private Member's Bill on the subject early in 1990.

The Law Commission has recommended the creation of three new offences

(i) Simple unauthorised access, punishable with a maximum of three months imprisonment or a fine
(ii) Unauthorised access, with the intent to commit or facilitate the commission of a serious crime punishable with a maximum penalty of five years imprisonment
(iii) Unauthorised modification of computer material punishable with a maximum penalty of up to five years imprisonment.

They have not addressed electronic eavesdropping or made changes to the Theft Act dealing with machine deception or made provisions governing the admissibility of computer evidence.

Users and vendors of information products/services alike are now facing a dilemma – How to improve the administrative and technical robustness of their solutions yet still maintain implementation momentum on a substantial information systems investment. The new Legislation may spawn a regulatory control framework that could demand changes to the way in which people interact with computer systems, access privileges are assigned and events are recorded and analysed (to use as evidence in criminal or civil court).

Good control to support legislation may add a substantial operational overhead in a distributed information system. This may be administrative as in human actions or technical as in the classes and types of hardware/software control devices to be applied (e.g. high speed Audit servers, Biometric authentication devices etc).

ICL has been considering the information security implications in a number of ways – from the overall architecture of information systems to specific components to support future regulatory frameworks.

This article is a preliminary review of the issues - Strategic 'generic' solutions to these problems will be fed into the product/service development cycle to ensure that ICL users can implement detailed control procedures.

Information Systems Architecture

Any generic design of access controls to secure I.T. products must consider the overall architectural direction of information systems. The strategy for 'computer' security must consider the increasing demands by society for systems to be simple to design, develop and use, as well as administered in a cost effective robust manner.

Security control barriers must recognise the needs of 'user friendly' Human Computer/Man Machine interfaces (HCI) and speedy, responsive connection to distributed information services Service Connection Management (SCM). An understanding of the architectural integration may give some valuable insight into the controls that could be put in to achieve adequate protection and how these might be able to support and reinforce the policing of systems, both for conformity with corporate security policy and for adherence to legislation concerning misuse of computers.

Any Legislation in enforcement must recognise difficulties in defining what constitutes a computer. It is increasingly an arbitrary set of packaged services necessary to support the processing/storage needs of an information system. It may be considered as one or more physical and logical objects within the prime management responsibility (domain) of an individual function, department or enterprise. The services offered by the 'computer' may be physically/logically discrete or be distributed around a worldwide enterprise. The computer may be housed in a portable token - the Smart-VLSI chip card - or may be a fixed unit embraced in a single building or site.

The Management of Security

An information system can consist of single or multiple management domains each with their own security policies over what constitutes authorised use.

The boundary between domains *may* go through one or more computers with/without human intervention. The level of computer functionality may range from a full, comprehensive end user computer to a single telecommunication switch.

The boundary of the domain is recognised in any access authorisation decisions; this may not always coincide in the context of human computer interface with any specific computing device.

Each management domain has a security policy. This applies to the domain contents, the use of the contents and the style in which the contents operate.

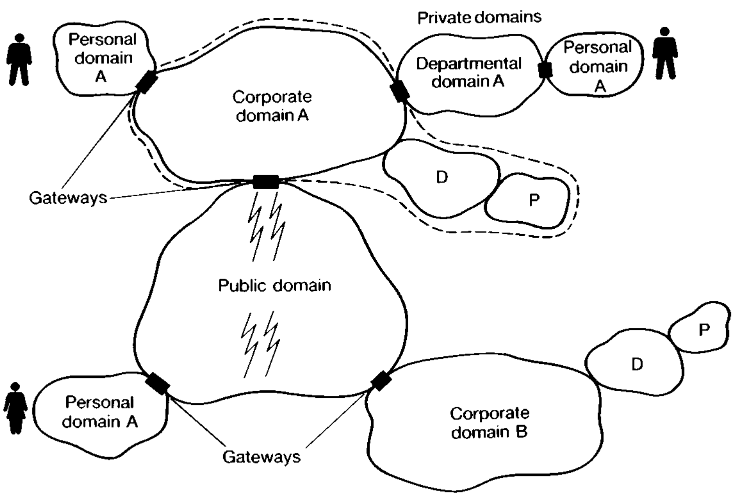


Fig. 1 Domain Control (logical

The formality and obstrusiveness of the policy is decided by (or on behalf of) the prime authority for that domain.

The Security Policy

The security policy may be considered as constituting three elements between the user and the information system (3 'A's) – Authentication (includes identification), Authorisation and Audit (both accountability and analysis).

The emphasis of how control is to be applied will vary by management domain and will require a balance of the 3 'A's to a lesser or greater extent. Some commercial user friendly systems will emphasise minimal authentication and authorisation but robust/tamperproof audit, others as in defence/-central government will dictate fundamental enforcement of access control/authorisation rules. Some domains will be publicly available and service suppliers will demand identification for subsequent charging of services (e.g. broadcast media, information bulletins, library services etc).

Any practical enforcement of legislation must recognise the boundary of the information system domain, and how far the 3 'A's of the security policy can be maintained at what level of attack. This would constitute the level of trust that any enterprise can have in its information systems.

The 3 'A's may be supported by physical/logical servers that can reside in different domains, the access control rules being applied not just between the information system (I.S.) domain and the user but between the I.S. domains themselves.

Access control

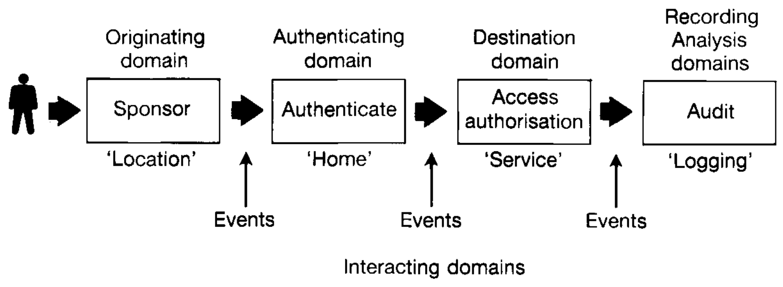


Fig. 2 Domain Relationships

The multiple domain nature of most information systems militates against the simple investigation and capture of any person perpetrating an unauthorised access. Current policing dictates that the 'criminal' be caught in the act – auditing of events must be in real time, there must be a prioritisation in the overall Network management (alert conditions) to ensure that incidents are routed to human interfaces for management/police actions.

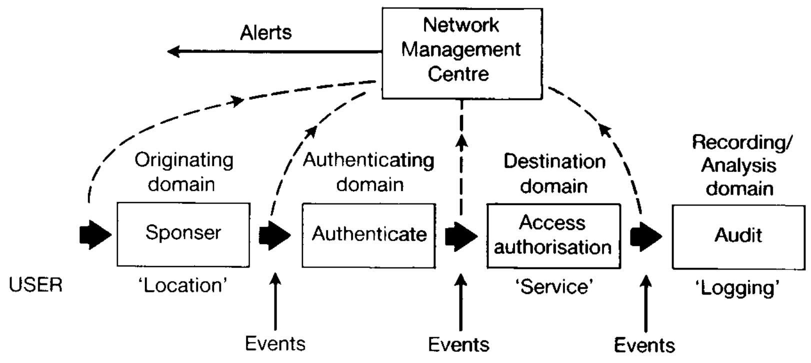


Fig. 3 Network Management

In highly secure systems, the Network Management process may abort the operation of certain gateway conditions between domains.

Architecture and Security

The enforcement of the security policy within an enterprise must be dependent upon the physical and logical positioning of the services within any information system, particular attention being given to the boundary between one domain and another where policing of access conditions is essential.

The ability of an information system to mandate and enforce the policing of its proprietary domains will depend upon the technical features of its products (primarily the functionality and assurance of its Operating Systems and Information Handling – Data Base and Data Directory etc). ICL is well placed to create robust solutions with its Secure VME and Secure UNIX offerings.

Three prescriptive architectures may be selected to illustrate the conditions, necessary to recognise legitimate use and withstand unauthorised 'hacking' attacks. The dates are general indications of by when comprehensive administrative solutions could be achieved by users.

1. *Organic* – the concept of a single security policy centrally administered, slowly embracing multiple tiers of managed I.T. resources (e.g. the Corporate Mainframe controls covering departmental and PC workstation units). This well disciplined approach from a clearly defined proprietary domain allows good central control over use and reduces hacking opportunities. It is the usual architecture implied when proposing legislation and enforcing hacking barriers, and audit evidence.

The Secure Corporate Mainframe (VME-High Security Option) is well suited to enforce a hierarchical 'bureaucratic' style of control necessary in sensitive, high risk information environments. It is currently prevalent in Defence/-Central Government.

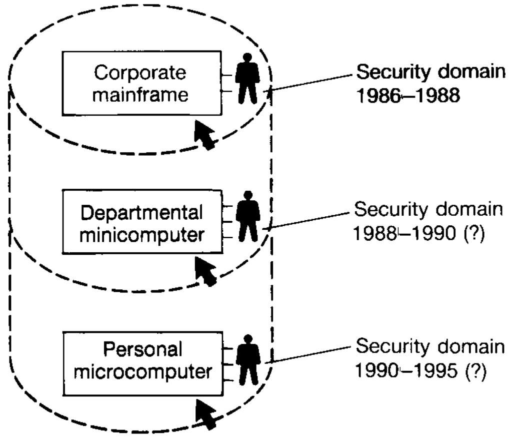

Corporate mainframe
Security domain 1986–1988
Departmental minicomputer
Security domain 1988–1990 (?)
Personal microcomputer
Security domain 1990–1995 (?)

Fig. 4 Architecture: three tier management, central resource control

2. *Zoning/Barriers.* The concept is of individual functions in an enterprise each with their own security policies (some weak, some strong) under a common human administrative encapsulation.

Access Management is hard to apply and hacking is difficult to recognise or stop, as control boundaries or barriers are administratively indistinct.

Examples of such a zoning/barrier approach may be found in those enterprises with multiple functions/lines of business locally supported by IT, but with functionally rich yet insecure centralised data processing facilities (e.g. many international financial institutions).

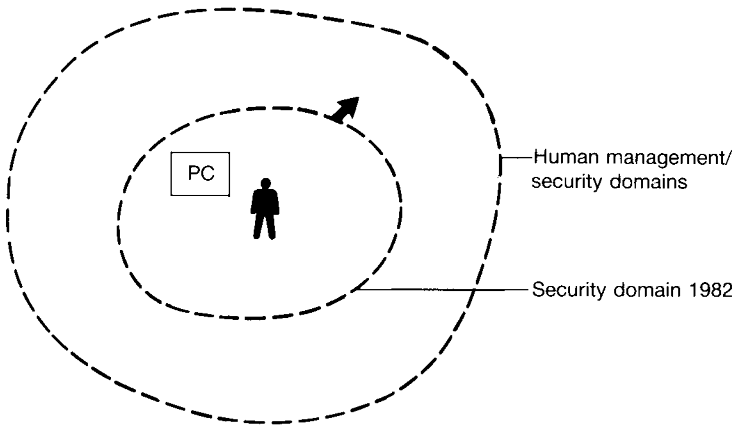


Fig. 5(a)Functional Control

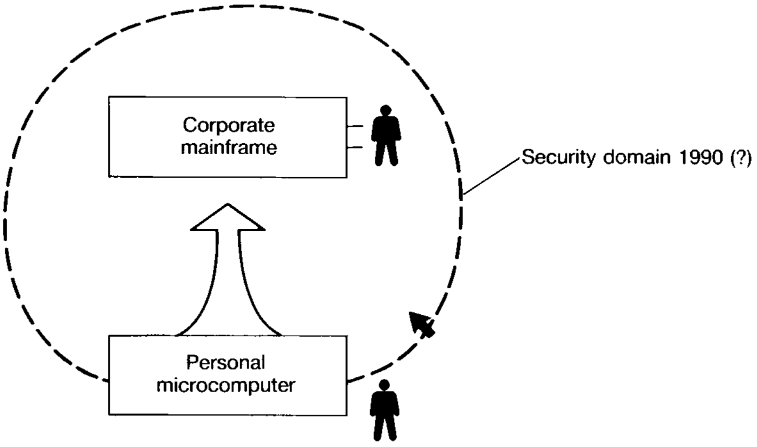


Fig. 5(b)Corporate Control

3. *Bonding*. The example taken is of a third party service supplier (e.g. PTT or added value Network Service – INS, SWIFT etc.) offering a security service embracing the external interfacing to, and communication between, two or more enterprises.

Many enterprises are small, physically diverse and/or highly dependent upon externally supplied I.T. operational services. These may consider that existing security policy, especially in the short term, is adequately enforced at the human administrative level and that disjoints in any I.T. controls are not considered to result in vulnerabilities against the perceived threats. The risk is usually in how the enterprises interface with each other over communication networks. Trusted, supportive services based on these external links may be an adequate response. The security policy of the user enterprise will reflect the current evolution of the externally available 'bonding' services.

Many of the earlier demands for legislation on unauthorised access and usage of information services have come from such service providers, who may lose substantial revenue as well as service credibility from external attacks.

In parallel they have been instrumental in developing the key components of secure interworking (reference the Open Systems effort in X.500 Directory and X.400 Message handling/mail services).

The boundaries between domains can be defined in commercial as well as technical terms, and are thus well defined. Policing to ensure only authorised use between domains will require robust gatewaying processes at a high level of operational assurance.

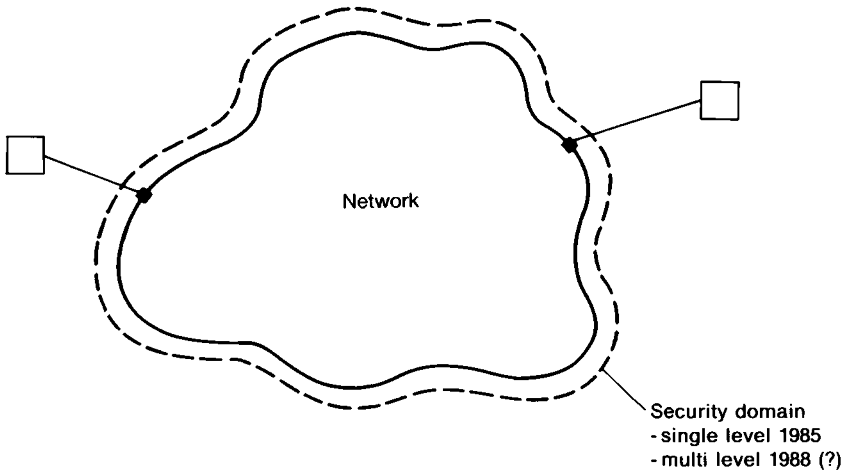


Fig. 6 Single tier management, network control

The typical larger enterprise will include examples of all three directions in the development and support of comprehensive security policies. Any convergence of the three approaches will result in a blurring of the control boundaries especially with the desired increase for publicly available services in the content of a more open society. Increasingly the information systems in any enterprise will lack strictly enforceable control boundaries unless rigorously enforced by corporate management.

Enforcing Access Control

Much of the debate surrounding the development of adequate legislation in the UK to counteract 'hacking' has revolved around the term 'unauthorised'. This implies that what is authorised or unauthorised at the boundary of any information systems domain is known to the user. This may not always be true. Poor human and/or system administration may compound the problem.

The information system may dynamically allocate access authorities to any potential user to do things (privileges) depending upon a series of both end-user and systems contextual issues (the identification, quality of authentication, the work station, the time of day). These may not be notified to the user in that role. In addition, every user of any information system may have privileges not known to him/her that he will only find out about when services are requested.

The problem may be analysed under the 3 'A's.

(a) *Authentication*

This must be considered to be more than just proving the identity of an individual human identity. It must be performed taking account of the organisation of society as a whole and of the specific enterprise concerned. Thus authentication must decide, for instance, between such alternatives as the following:

- actual human *actions* and *adventitious effects* on the system due to natural causes or equipment failure,
- the *identities* of different individuals,
- the different *roles* the same person may play interacting with a system, where each role may carry different privileges and responsibilities, and
- the specific *tasks* an individual may be assigned either permanently or temporarily to achieve the objectives of that role.

Such checking can be compound as in recognition of individuals with multiple roles in an organisation (e.g. both public society and private enterprise), using a combination of criteria.

The individual may not know that he/she is being authenticated (covert), the control barrier of the information systems (proprietary, public) or the level of privileges that are/could be assigned. The authentication products must reflect the overall control philosophy in addition to just proving the identity of the individual from biometric or other means.

(b) *Authorisation*

The access authorities given to individuals and groups of individuals must be clearly stated, agreed and enforced.

In much discussion on legislation, authorisation implies that the role, and hence privileges, of an individual in an enterprise have been pre-assigned and are known to the individual, not whether the role is consequential to the identity and authentication of the human.

Increasingly the system contextual nature of control will condition the privileges and tasks that can be carried out by an individual (e.g. a specific terminal must mean Systems Administrator).

Authorisation/access control to any managed information systems domain must ensure a well defined boundary with suitable controls. This may mean integrated, 'Warning banners/No Trespass signs' on a machine, as a fundamental element of good man-machine relationships (The ergonomics of such apparent barriers as in the impact on ease of use for authorised individuals needs to be more fully explored). Such barriers when breached would trigger events that would be used for subsequent analysis.

It is currently difficult to achieve a common understanding of boundaries based on controls that are primarily driven by identity-based authentication and the supporting mechanism. This may be reflected in the poor record of convictions for hacking, under existing legislation.

- Does just taking the identity of someone else (masquerading) constitute a crime, if there is no obvious gain etc on the part of the perpetrator?
- Does a user-friendly Man Machine Interface 'Welcome to the Data Centre' constitute an invitation to hack the information system etc etc?

The judicial system has been benevolent in interpreting current weaknesses in systems design to the benefit of the plaintiff.

(c) *Audit*

In many enterprises, the role of audit may be paramount in defining the level of control expected and the Services volume of supported data to be collected with the system. A good information system will require

- review of patterns of access to objects by individuals/processes.
- discovery of (*repeated*) attempts to bypass protection mechanisms.
- use of privilege functionality greater than that 'authorised'.
- deterrent against (*habitual*) attempts to bypass systems protection.
- user assurance that records are being kept of attempts to bypass machines.

In this list, primarily derived from the US Authorities – Guide to the Understanding Audit in Trusted Systems, emphasis is placed on protection mechanisms and multiple (yet undefined) numbers of attempts to bypass them. The difficulty of enforcing differing 'threshold levels' by industry, discipline, or application is obvious especially in the context of liberally interpreted legislation.

A more fundamental view of the boundaries to information systems, their associated domain barriers based on the perspective of the service user may be necessary for the construction of suitable legislation and products to enforce the controls.

The Control of Privilege

The emphasis on control over information systems access must reflect the fundamental nature of Security Policy, which is the overlap of the Organisational resources (people, roles, task etc) with the Information Systems resources to be protected. If there is no overlap there is no need for control.

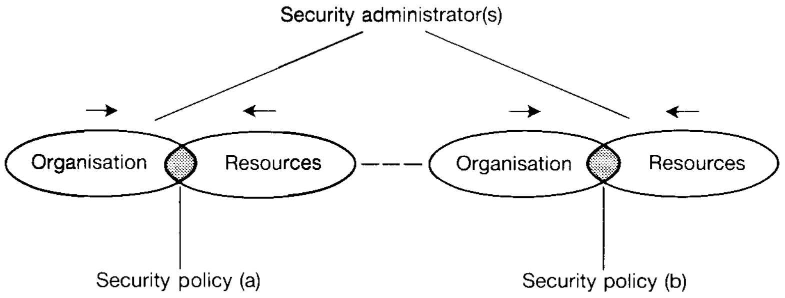


Fig. 7 Administration of Security Policy

Hardware/software control systems have primarily been developed by I.T. service suppliers who have a 'resource' contextual view of the world. The users, assisted by advisors (Auditors etc) have primarily approached control from the organisation viewpoint.

Current debate on legislation over access to systems must reflect administrative solutions that converge the two approaches.

Vendors, including ICL, can support this process by developing and implementing a common architectural approach to (distributed) information systems.

A common information system boundary of access control policing based on asking for and/or obtaining privilege could have an architectural simplicity. This could create a universal control point in any system that could be consistent for all potential users, whether internal or external, whether authorised or unauthorised. Those that asked for, obtained or accepted privileges beyond their agreed level would be carefully monitored.

The monitoring of privilege request or allocation post-ID/authentication is considered a more robust yet selective auditable process especially in the establishment of likely 'hacking' controls. The evidence relating to access or attempted access would be more selective and yet more likely to show some subsequent malicious intent.

A common philosophy would allow the unifying of technical solutions covering all types of 'privilege packages' in the form of VLSI 'Smart Cards' through to dynamic, run time, privilege certificates obtained from central I.T. sources. It could allow a single control barrier around any number of domains in a multi-domain information system.

A suitable positioning of the control boundary *after* the simple authentication stage would also reduce the problems associated with such large volumes of audit records of those events (for use as evidence both internally and, possibly, externally for civil litigation or criminal prosecution). It would also permit a common method of management of the records created by many of the authorisation controls for both internal and external "attackers".

The Future

ICL, in collaboration with other I.T. service suppliers, is currently exploring the various architectural approaches, especially within distributed office information systems, to ensure adequate user access control systems.

Much of the security work is in the public domain in the standards bodies of ISO, CCITT and ECMA. Particularly relevant is the ECMA TC32 TG9 work on security, with its understanding of distributed secure services to support productive applications executed on behalf of the end users. The concept is of locally trusted User Sponsors, representing the user, communicating with both productive applications and supportive systems' contextual services. These use high integrity Privilege Attribute Certificates (PAC's), to establish both good organisational definitions (identities/roles) as well as creating effective boundary conditions between information systems domains.

ICL with its partners in Bull and Siemens is prototyping many of these concepts in an EEC funded R&D programme – SESAME. Early trial facilities are currently being demonstrated. Deliverable products having this capability are expected in the early 1990's.

Access control over information systems is a combination of legislation, and effective policing and control information systems products.

Legislation to be enforceable must recognise the changes in society brought about by the explosion of information services both public and proprietary, and the associated control solutions.

A society circumscribed by controls over information must allow some flexibility in what constitutes a computer access violation. The ITT&B industries in return must ensure that information systems' architectures, services and products do reflect and support legislation.

NO criminal must ever go free because of the successful plea – there were no controls so I though I could do it!

References

Law Commission (89) - On Computer Misuse W.P. 110 (and associated working papers).
ECMA (88) TC32 TG9 - TR46 - Security in Open Systems – A Security Framework.
IEEE (88) - The Source of Authority for Commercial Access Control. J. Moffett/M. Sloman.
UPDATE (89) – An Abuse of Privilege – C. W. Blatchford (Volume 2.1–9/89).
MILCOMP (88) – Management and Security – Six Principles. C. W. Blatchford.
D.O.D./NSA-NCSC - 'Rainbow Books' and associated Implementation Guidelines. (ref. A Guide to Understanding Audit in Trusted Systems 7/89).

Standards for secure interfaces to distributed applications

T. A. Parker
ICL Defence Systems, Winnersh, Wokingham, Berkshire, UK

Abstract

The large distributed systems of the present day have users who access many different applications residing in different end-systems supplied by different vendors. The identity and access rights of these users need to be established, communicated and managed in a way which enables the disparate components involved to interwork in a secure manner. This paper describes work underway in ICL and the international standards arena to develop standards within which this can be achieved.

1 Background

This paper describes one topic of work recently completed within the European Computer Manufacturers Association (ECMA). The area covered deals with the topics of authentication and access control. The work was done by ECMA/TC32-TG9, which is the ECMA task group responsible for the definition of standards for security in open systems. The paper represents the author's own interpretation of the work, and should not be taken as a definitive statement of an ECMA position.

The TG9 group started work in 1987 as a result of a joint initiative by ICL in the UK and NCR in the Netherlands. At that time, although work was being done within ISO on specific layered communications security services, there was no standards activity looking into the security information that needed to be communicated between standard applications for them to interwork in a secure manner. Within ECMA, the Task Group working on distributed office applications (ECMA/TC32-TG5) had specific requirements for expertise in this area, and one of TG9's first tasks was to provide them with the necessary support. The scope of TG9's activities was later widened to cover applications of all kinds.

The first ECMA Technical Report from TG9 [4] was produced in 1988. It gave an abstract framework within which security interworking standards could subsequently be defined; this work was reported in [1], [6] and [7]. The next phase of the work is now complete, and is documented in an

ECMA Standard [3]. It gives the syntax and some semantics of the data elements which need to be exchanged, and outlines the security services which are appropriate candidates for further standardisation work.

Although the work embraces all aspects of security, including for example audit, interdomain working and recovery from security attack, the most critical and pervasive areas are those of authentication and access control. These topics are addressed in this paper.

ICL is actively supporting this architectural standardisation work. Its own internal company strategy [8] for security closely follows the precepts laid down in this paper and in the standards documents referenced from it. Products and prototypes are being developed within the company to conform to standards based on this architecture.

2 Definition of terms

There follows a short list of the main security terms used in this paper. The definitions are informal and are intended as an aid to understanding. They are compatible with the definitions and explanations given in the formal ECMA documents.

Control Attributes	Attributes associated with a Security Object used in conjunction with Privilege Attributes to control access to that object. Examples are: an access control list, a security classification.
Privilege Attributes	Attributes associated with a Security Subject which, when used together with the Control Attributes of a Security Object being accessed, define that Subject's access rights with respect to that object. Examples are: the Subject's identity, a security clearance, a group membership.
Privilege Attribute Certificate	A collection of Privilege Attributes and information controlling their use, bound together under a cryptographic seal.
Security Object	An entity to which access is controlled under a security policy. Examples are: a workstation, an application, a file, a data item.
Security Subject	An entity in active mode requesting access to a Security Object. A security subject can be compounded from a number of subject components, each of which contributes to the compound subject's Privilege Attributes. Examples are: a user, an application, a workstation, a user and workstation in combination.

3 Requirements

Application standards are being defined for applications which are to be usable and manageable in the context of large distributed systems supporting multiple applications of different types. Such standards must therefore be developed to take account of this context.

As users of computer systems become more sophisticated, and as more activities in business begin to involve computers, each individual user will typically need to use an increasing number of applications, possibly in parallel. Also, the emergence of a variety of access control models based on access privileges other than simple user identity, coupled with the need in large distributed systems to manage these privileges independently of particular applications has led to a requirement to enrich a user's identity with additional data which defines these privileges.

The ECMA Security Framework [4] offers an approach which permits a user to authenticate himself only once to a system, to obtain as a result a certified collection of access privileges, and to use these with a number of different applications. These privileges are represented in the Framework by Privilege Attributes associated with the user; the user's identity may be one such attribute. Also, like users, applications themselves can be authenticated, and may possess Privilege Attributes to be used when accessing other applications; this will be elaborated in the next Section. In the ECMA Standard [3], developers of application standards are provided with a common mechanism for offering and accepting these values. The data construct central to this is the Privilege Attribute Certificate (PAC).

The next Section outlines some of the basic architectural principles upon which use of the PAC is based; this is followed by a Section giving a more detailed description of the internal contents of the PAC.

4 Architectural overview

If a Security Subject (e.g. a human user) is to authenticate himself only once to a distributed system, the points in the system at which access control decisions are made will not necessarily be co-located with the point of authentication. Therefore there must be some mechanism by which any proof of identity obtained as a result of authentication is subsequently propagated to the different access authorisation functions residing in the various End-systems of the distributed system. In the ECMA Standard, a certificate sealed by an Authority acceptable to the access authorisation functions of the system is used for this purpose.

Under many access control policies, a Security Subject's access rights are not based simply on the Subject's identity, but on attributes that the Subject possesses such as organisational role, or security clearance. These are the Privilege Attributes introduced in the previous Section. Attributes of this kind are directly related to the Subject, not to the Objects that the Subject may wish to access, and they should therefore be handled and managed in relation to the Subject. Indeed from an access control point of view, the Subject's identity is just another Privilege Attribute (it is only when accountability comes into the picture that any distinction has to be made). For these reasons, the certificate that is produced for an authenticated Subject contains all of that Subject's Privilege Attributes (or at least all that the Subject asked for). This certificate is known as a Privilege Attribute Certificate, or PAC.

Before a PAC can be created and issued, the Subject must be authenticated. The security functionality which ensures that this is done, and which subsequently orchestrates all of the Subject's security related interactions with the distributed system, is known as the Subject Sponsor. The scenario in outline is as follows:

The Subject Sponsor first directs an unknown Subject to an Authentication Service which engages in a standard protocol with the Subject Sponsor to authenticate the Subject, the Sponsor simultaneously interacting with the Subject as required to obtain the necessary authentication information. When the subject has been successfully authenticated, a certificate containing the Subject's identity is generated by the Authentication Service. This certificate is passed to an Attribute Service which recognises the Authority of the Authentication Service. The Attribute Service then attaches the Subject's other Privilege Attributes, adds various items of control information and seals the package; this package is the Subject's PAC.

It should be noted that although the authentication and attribute handling functions are architecturally separate entities, they will commonly be implemented as one combined security service.

The sealed PAC is then returned to the Subject's Sponsor for distribution to end-systems and their applications, as selected by the Subject.

5 PAC contents

Privilege Attributes are particularly susceptible to misuse because they are frequently under the control of software entities which cannot be trusted not to tamper with them. They may therefore need a variety of forms of protection as shown below:

- Protection against undetected modification
- Protection against use by the wrong Subject
- Protection against use outside stated constraints (for example a PAC may be issued for use only with respect to a particular end-system or application, or may be valid only for a particular time period)

To provide these kinds of protection, a variety of optional control fields have been defined for the PAC. For a detailed description and explanation of these and other fields within the PAC, the reader is referred to [3]. The contents of a PAC are, however, outlined below:

ASN. 1 Field Name	*Description*
syntaxVersion	For future-proofing purposes
containedPACs	For combining PACs produced by different Authorities
privilegeAttributes	The actual Privilege Attributes being carried
initiatorQualifierAttributes	Attributes that must be associated with the submitter of the PAC for the PAC to be acceptable
targetQualifierAttributes	Attributes that must be associated with the access authorising function with which the PAC is to be used
validationKeyIdentifier	The identifier of a cryptographic key that can be used in a variety of ways to protect the PAC against misuse
creationTime validityTime	Fields controlling PAC expiry
pacIdentifier recoveryIdentifier	Fields used for PAC usage monitoring and revocation
auditIdentifier	Used for accountability purposes when the identity being used for access purposes is either not present or is different from the Subject's true identity (e.g. someone acting for a person who is on leave)
chargingIdentifier	For allocation of resource use charges
pacAuthority	Identifies the Authority that sealed the PAC
pacSeal	The PAC's cryptographic seal

Of these fields, the only ones that are not optional are the syntax version, the Privilege Attributes themselves, and the Authority and Seal.

6 Standard attribute types

Like other kinds of attributes, Privilege and Control Attributes can be standardised, and [3] identifies a candidate selection. Since these security attributes will often be handled in a similar manner to other attributes, it is useful to define them with the same general syntax. However there is a difference between security and other attribute types which requires additional internal structuring of the attribute value. The difference is due to the fact that different Defining Authorities may be involved in defining security attributes of the same general type. The value field is structured therefore into two parts, an Authority and a (true) value. Notice that a Defining Authority is different in kind from a PAC issuing authority, the former being a human administrative body, the latter being a software entity. To help highlight this distinction in the text below, the name of the former is capitalised.

One use of the Authority field arises because in a particular security domain there may be defined a security policy which demands that multiple sets of a given attribute type be supported, each set belonging to a different Defining Authority, and being used to support a different aspect of the policy. This means that applications must be able to distinguish between attributes of the same type belonging to different sets. The Authority field enables this distinction to be made. An example might be a system supporting two label hierarchies belonging to two different national governments. From an implementation point of view an application that supports two hierarchies is not concerned (apart perhaps from some simple name-value mapping tables) with what authorities they belong to but only that there are two distinguishable hierarchies.

A second use arises because some attribute types are related in that they have complementary functions within the same security policy. For example different types are used to specify upper and lower bounds of the same kind of access privilege. By separating type from Authority an application can more easily see the correspondence.

Finally, by separating type from Authority it is possible for an application developer to define the security attributes it supports, without being aware of the potentially long and certainly dynamic list of possible Defining Authorities whose identities would otherwise have to be bound into the type field.

7 Impact on application standards developers

The rest of this paper examines how adoption of the ideas developed above would affect the thinking of people developing international application standards. It first looks at how a standard application might receive PAC information, it then analyses the complications introduced by the possibility of applications acting by proxy for their users (which may be other applications), and finally gives some guidelines on how to define application standards in a way which maximises their independence from the different security policy regimes within which they may be required to operate.

7.1 Delivering PAC information to applications

External support for providing a target application with the attribute values in its accessing subjects' PACs can be given in two ways:

1 The application can depend on a co-located trusted infrastructure (for example its host operating system, or in the future perhaps the ACSE

layer) to provide them. In the ECMA Framework, the infrastructure will have obtained the attributes in a PAC, via a Secure Association Service in a manner similar to that described below for the application itself. It will have checked the validity of the PAC, will have unpacked it, and may also have performed some mapping of the Privilege Attributes found in the PAC, into locally understood, possibly application-specific values. This mapping is an example of a mapping between the global system-wide Security Domain and the Sub-domain of the local End-system.

2 The application can obtain the identity of the accessing subject and other privilege attributes directly via a PAC which is certified by a (usually) remote authority.

In the first case, application access protocols are affected only in that no explicit authentication is visible; application standards designers should therefore permit this case as an option. Standardisation of the interface between the application and its host infrastructure is primarily an application portability issue, not an interworking standards issue. Standardisation of the relationship between an application and its OSI communications interface in respect of receipt of authentication data is an OSI standardisation matter.

Notice that the first case is useful for applications which were designed before the framework had been developed, and which previously depended on infrastructure-provided security controls and information to preserve their security. Using this approach it is possible for the infrastructure to conceal from the application any knowledge of the PAC itself, but instead to establish a working context for the application unchanged from its familiar one. Although this does not exploit the full power of the ECMA model, it nevertheless helps considerably in the integration of existing applications into the overall architecture.

In the second case, the PAC may be passed on any of three occasions, depending on the control relationship between the accessing subject and the communication association Bind operation. They are as follows:

1 When each accessing Subject connects to the application via an association which is dedicated to that Subject's use, a PAC representing some or all of the Subject's access rights is passed as a Bind argument (PAC A in Fig. 1).

2 When a single association is multiplexed between different accessing Subjects (for example in some transaction processing orientated environments), then if there is a concept, probably application-type specific, of a within-association 'sub-connection' the Subject's PAC will be passed as an argument to the 'sub- connect' operation (PAC's B1 and B2 in Figure 1); otherwise it can be passed with every operation, as in the connectionless relationship next described.

3 When a connectionless mode of access to the application is supported, a

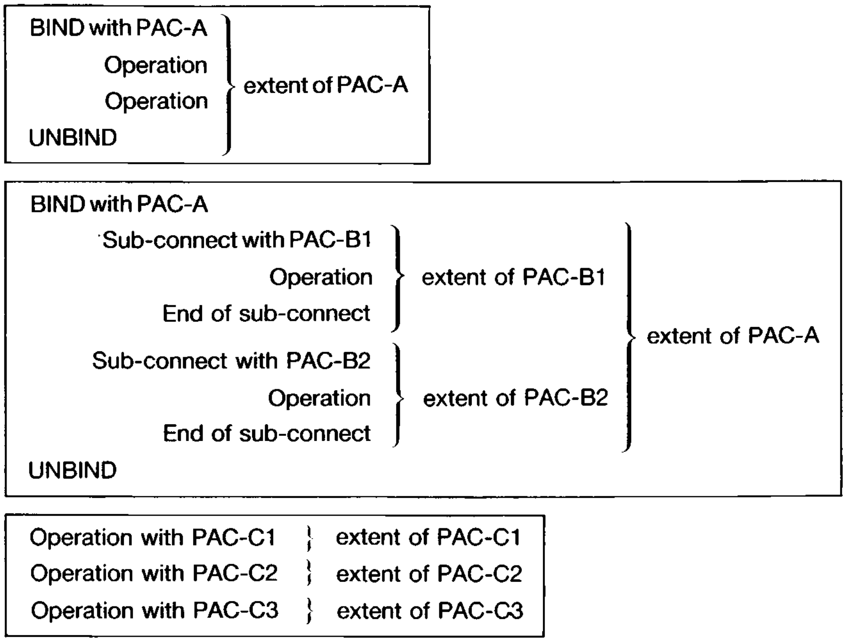


Fig. 1

PAC is presented with each operational command (PAC's C1, C2 and C3 in Figure 1).

The use of all three methods in combination is possible.

7.1.1 Specification in an application standard The specification of an association Bind operation should be structured so that application-specific arguments are easily separable from common arguments. The security-related common arguments identified so far for a BIND operation are:

- authentication information presented to enable the application to authenticate the accessing subject (this is not conformant to the ECMA view but may be required for systems in which no external support for authentication is provided)
- a PAC to establish the access privileges for operations performed in the context of the BIND. This assumes external authentication by a trusted authority has already taken place;
- nothing, assuming the infrastructure support described above.

The specification of a 'sub-connect' operation should include the optional presentation of a PAC to establish access privileges for operations performed in the context of the sub-connection.

The specification of other kinds of operation should include the optional presentation of a PAC containing attributes required specifically for the operation.

7.2 Proxy and compound subjects

Some operations requested of an application are not performed directly or completely by that application, but are wholly or partially 'sub-contracted' out to another application; for example a text editing application may use a file server to hold files of text; when a user asks to start an edit, the editor must ask the file server for the user's file. This is a case of Proxy; the editor is acting with respect to the file server as the user's proxy, and it must convince the file server of its right to access the file. Depending on the trust relationship between the two applications (which is determined by the security policy in force); it may do this in a number of ways:

1 The editor may be trusted only to ask for files that the user may legitimately edit. In this case the editor acts as the only accessing subject; the file server sees only the editor's access rights and expects a PAC from the editor, which belongs to the editor, and which is presented in one of the ways described in the previous clause.
2 The editor may have no access rights of its own to the files in the file server, and must prove to the file server that it has permission from the user to obtain the file. It does this by passing a Proxy PAC to the file server as a parameter to the access request for the file. The Proxy PAC belongs to the user, and was obtained from the user parametrically either with the user's operation request or via a previous operation. The proxy PAC may have been constructed to constrain tightly the editor's access (for example it may permit read access only to a particular named file)
3 As above, the editor has no rights of its own but a Referenced Data Transfer (RDT) facility [2] is supported by the file server. To perform a Referenced Data Transfer the user warns the file server in advance that the editor will be making the access request, and as a result obtains an RDT Reference from it. The RDT Reference is a tailor-made PAC authorising the transfer operation; it is sealed by the file server itself and is therefore usable only at the file server, and only for the operation specified. The user passes the RDT reference to the editor as a form of Proxy PAC and the editor uses it in its subsequent call on the file server as an operation PAC. The syntax of an RDT Reference was developed before the TC32-TG9 work matured and is at present special to the RDT Standard [2] but it is hoped that it will in future be possible to bring it and the PAC syntax in line.
4 The security policy in force takes into account both the access rights of the user and the access rights of the editor (for example the user may be permitted access to the file only if it uses the editor to access it, and the editor has access only if it is acting for the user). In this case a combination of 1. and 2. above occurs. The file server is being accessed by a Compound Subject.

7.2.1 Privileges required to form a bind When an association is being formed between two applications on behalf of a user, the access privileges required in order to make the BIND succeed may be different from those which will be established for operations conducted over the association. In particular, the user may not himself possess sufficient authority to form the association, and the relevant authorisation logic may require an additional PAC (e.g. from the initiating application) to be passed to it as a BIND argument.

7.2.2 Specification in an application standard The application standard writer cannot assume any particular trust relationship between two applications, since this will vary according to security policy. Therefore:

- for operations that may be devolved to a second application, the first application should be capable of accepting a Proxy PAC from the originator of the request, for use by it as the required operational PAC in its operation call on the second application;
- when an (initiator) application binds to another (target) application it must be able to pass a PAC representing the access privileges required to permit the establishment of the association, as well as one for use over the association.

7.3 Secure application associations

[3] defines a Secure Association Service which provides for secure communications between applications; this needs to be informed of the nature of the OSI communications security services wanted. The information must be passed when the association between the applications is formed at Bind time. Other responsibilities of the Secure Association Service do not impact on the specification of application standards.

7.4 Example ASN. 1

The following ASN. 1 is offered as an example of changes to interface definitions required to support a common application access control framework.

```
Bind- Argument ::= SEQUENCE {
      application-specific-argument  [0] App.
      common-security-argument       [1] Sec OPTIONAL}
-     App ::= whatever the application requires -
      Sec ::=  SEQUENCE {
                 CHOICE {
                 credentials-for-appl [0] Creds OPTIONAL,
                 certified-id-and-privileges [1] Bind-PACs OPTIONAL},
                 bind-security ServicesRequired OPTIONAL}
```

```
        Bind-PACs::= SEQUENCE SIZE (1...2) {
                     required-bind-PAC [0]
                         PrivilegeAttributeCertificate OPTIONAL,
                     context-PAC [1]
                         PrivilegeAttributeCertificate OPTIONAL}
  -     PrivilegeAttributeCertificate as defined in [3] -

  Sub-Connect-Argument ::=      SEQUENCE{
                                sub-connect-specific-argument [0] Sub,
                                context-PAC [1]
                                    PrivilegeAttributeCertificate OPTIONAL}
  -     Sub ::= whatever the sub-connect requires -

  Operation-X-Argument ::= SEQUENCE{
                           operation-specific-argument [0] Op,
                           common-security-argument [1] Op-Sec}
  -     Op ::=      whatever the operation requires -
        Op-Sec ::= SEQUENCE{
                   operation-PAC [0] Operation-PAC OPTIONAL,
                   proxy-PAC [1] Proxy-PAC OPTIONAL}
```

Notes

1. credentials-for-appl is used to pass actual authentication information, for the target application itself to authenticate the accessing subject. This is appropriate only for use in systems not conformant to the ECMA Framework.
2. bind-security is used to tell the Secure Association Management Service what kind of security protection is needed on the association that will result from the Bind.
3. required-bind-PAC is the PAC used to present the privilege attributes required to perform the Bind.
4. context-PAC is the PAC used to present the privilege attributes to be associated with operations to be performed in the context of the BIND or sub-connection.
5. required-operation-PAC is the PAC passed with an operation when the access privileges currently established are insufficient to permit the requested operation. This may happen when either the operation requires special additional privileges, or when a Bind is being multiplexed between multiple subjects.
6. proxy-PAC is used when the target application needs to make further accesses to another application on behalf of the calling subject, and itself would otherwise have insufficient access rights. The target application will itself then use this PAC as an operation-PAC when it acts as an initiator application in its call on the other (now the target) application.

7.5 Object security attributes

When an application object is created, the security control attributes that are used to protect it from unauthorised access need to be defined and established. The exact way in which this is done will be particular to the security policy in force, but under some policies some of these attributes will be specifiable as arguments to the 'Create' operation. When the object is accessed, its control attributes may need to be read or written. The syntax of a security attribute is the same as other kinds of object attribute and an application need make no distinction between the two kinds, except in the following respects:

- the standards developer should clearly separate out any rules that may be specified about what security attribute values are acceptable in what contexts, so that the security policies supported are in this respect clearly identifiable;
- these rules should be specified as optional to allow for variations so that the application can be used in different security policy contexts;
- the definition of the types of control attribute that are to be supported should be similarly flexible and easily separable;
- the standards developer should consider the advantages of separating out operation arguments containing security attributes from other arguments to simplify any functionality that an implementer might provide to protect them and control their use.

7.6 Access authorisation

An application is responsible for authorising access only to its own objects, though it may use its host infrastructure to support it in doing this. It may optionally choose to control access to itself, though in many environments this control is provided externally. The authorisation logic needs information on which to base its decisions; these are the privilege and control attributes described extensively elsewhere, coupled with information about the access context pertaining at the time of the access requests.

An application standards developer defines the access controls that the application is required to support. However this definition will always constrain the security environments within which the application will be able to be used, and the definition should therefore be arranged to be optional and separable from the rest of the standard definition so that variants appropriate to different security policy environments can be defined and developed. In due course commonly used access control policies will be profiled. This is an area for further study.

8 Further work

TC32-TG9, (with the author as document editor), is now turning its attention to the detailed definition of a standard protocol to a combined Authentication and Security Attribute Service, by means of which users authenticate themselves and obtain PACs. Meanwhile testing of the practical feasibility of the approach is underway within a number of the companies involved in the work, including of course ICL, and intensive liaison work continues with ISO and CCITT, where the ideas behind the ECMA work are beginning to appear in draft standards, for example in [5].

9 Conclusions

The ECMA work has resulted in a framework within which access control information can be generated and distributed in standard ways. These permit existing systems to be evolved gradually to work within the framework; in particular, applications developed outside the context of the framework can benefit in part from it.The framework enables system users to authenticate themselves once to the distributed system as a whole, and supports a wide variety of access control policies. The framework is based on sound, practical implementation considerations which are continually being tested by experimental and product design and development.

References

1 COLE, R. Hewlett Packard, Bristol 'A Model for Security in Distributed Systems', submitted for publication in Computers and Security magazine.
2 ECMA STANDARD ECMA-131 'Referenced Data Transfer'.
3 ECMA STANDARD ECMA-138 'Data Elements and Service Definitions' December 1989.
4 ECMA TR/46. 'Security in Open Systems – A Security Framework', July 1988.
5 ISO/IEC JTC1/SC21/WG1/F29 'Working Draft Access Control Framework', Florence meeting November 1989.
6 KRUYS, J. P. 'Computer Security and Open Systems', Computers and Security, 8, pp 139–47, Elsevier, April 1989.
7 PARKER, T. A. 'Security in Open Systems – A Report on the Work of ECMA's TC32-TG9'. Proceedings of the 10th National Security Conference, Baltimore. September 1987.
8 PARKER, T. A. 'NPL Networked Security Architecture', ICL Product Specification Document PSD 2675 Issue 2, 8th March 1988.

How to Use Colour in Displays – I. Physiology, Physics & Perception

Darren Van Laar and Richard Flavell
The Management School, Imperial College, London

Abstract

Colour is often used in visual displays to improve human-computer interactions and achieve a more natural and representative display medium than monochrome coding alone can accomplish. In order to use colour on displays effectively it is important to understand how colour behaves as a physical phenomenon, how VDUs display colour, how people perceive colour and how colour can be used to encode displayed information.

This is the first of two papers discussing the effective use of colours in CRT displays and covers how colours are produced and measured, the physiological basis of human colour vision and the perceptual effects which can be expected when using colour on computer displays.

The second paper will discuss how to increase the effectiveness of displays by using colour to link displayed information with the user's tasks. The different ways of applying colour to displays, and the advantages and disadvantages of each will also be described. The paper concludes with a summary of the physical and cognitive factors that have been highlighted by the two papers, together with some guidance as to how this advice may be implemented.

1 Introduction

Employing colour in displays is a task fraught with difficulties. The naïve programmer or designer will write a program or design a system and then decide to make the display 'look more interesting' by adding some colour. But already questions arise: which colours to add, how many colours to use, where to use them, how to make them look good in combination, how to find a particular colour, how to ensure the legibility of colour text and so on. Ten, even five years ago answers to such questions were easier to find as many of the options were limited by the technology of the display systems available. That is, if your display only has 4 or 8 colours, then it is easy enough to assign colours to everything in sight and stop when you run out. But with 16, 256, and 16 million colour systems being widely used today, merely choosing between the colours becomes a problem let alone deciding which parts of the display ought to be colour coded, or considering how colour should support the task in hand.

This article will attempt to overcome such problems by explaining basic psychological and physical colour phenomena clearly and in simple terms. This is necessary because many people who might gain benefits from effectively employing colour are put off either by lists of unjustified guidelines, or by jargonised and incomprehensible physics, physiological and psychological text books.

This, the first paper of two looking at the use of colour on computer displays, will present the background necessary to understand how to employ colour, and will answer questions such as what are colour spaces, how can displays be designed to take account of colour blindness, what are the best ways to describe and think about colour, what makes some colours easier to see than others, and how do visual displays produce colour. The second paper will deal with the more psychological and cognitive aspects of colour use, discussing instances of where colour has helped in computing tasks, how to allocate colour to displays, how many colours to use in a display, how to make colour displays aesthetically pleasing and how to derive links between colour coding and the tasks it supports. At the end of these two papers the reader should be able to make much more sophisticated use of multi-colour displays, be able to understand colour terminology and concepts that human factors and computer graphics designers use, and be able to produce more informative, aesthetically pleasing and productive displays than before.

2 How do people sense colour?

The difference between sensing something and perceiving something can be quite subtle. This section will look at how light is sensed by the human visual system, and what this tells us about how colours are seen.

2.1 The spectrum

All modern theories of colour and colour vision stem from the discovery by Sir Isaac Newton that white light could be split by a prism into a 'spectrum' of colours and them recombined back into white light. Newton realised that as light passes through a prism the amount it is bent by the prism (refracted) depends on the wavelength of the light. The fact that we see this spread of light as differing in hue implies that the way we see colours depends on the wavelength of the light.

Visible light is a small part of the entire electromagnetic radiation spectrum which includes other forms of radiation such as X-rays (at the short wavelength end) and radio waves (at the longer wavelengths). The colours we see in the range of visible light gradually merge with one another as they change from the violet at a wavelength of about 380 nanometres (nm,

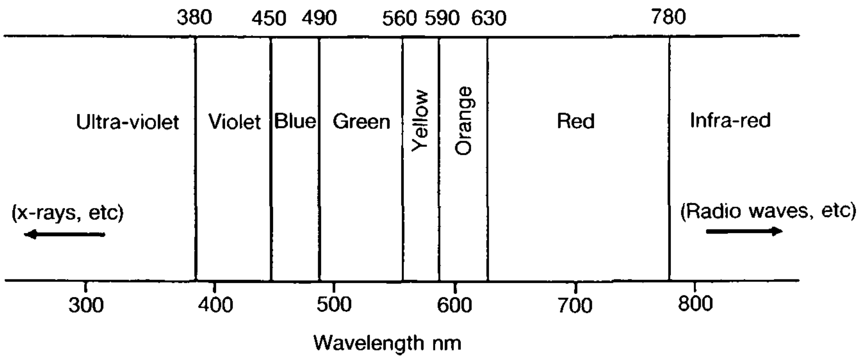


Fig. 1 Part of the electromagnetic spectrum, showing colour names usually associated with such areas by wavelength.

millionths of a millimetre) through the familiar rainbow colours to about 780 nm where red merges with the infra-red. Figure 1 shows the portion of the electro-magnetic spectrum from which visible light comes, along with common colour names typically associated with regions of the spectrum by wavelength. An important point to remember is that not all of the colours we can see are represented along the spectrum: purple is a mixture of red and blue light, and browns are actually dark yellows and oranges.

2.2 Basic physiology of colour vision

As light enters the eye it passes through a transparent protective covering (cornea), a fluid filled cavity is focused by a crystalline lens, passes through another fluid filled cavity, past layers of nerve cells and assorted internal wiring, through a light sensitive layer (retina), and finally is absorbed by a non-reflecting coating on the back of the eye (see Fig. 2).

The light sensitive layer consists of over 125 million cells of which approximately 95% are light intensity sensing 'rods', and 5% are colour sensitive 'cones' (so-called because of their shape). The light sensitive cells are unevenly spread across the eye, with colour sensitive cones being most intensely concentrated at the point where light and images are focused at the back of the eye (the eye's visual axis) at a point called the 'fovea'. Few, if any, rods occur at the fovea, and it is the area of the eye with the greatest concentration of cones and where the finest detail and colour discrimination occurs. From the fovea cones are present to within about 50° of the eye's axis, after this point only rods are present and only shades of grey are perceived.

Note that in the vision literature the size of objects is often measured in terms of degrees and minutes of 'visual angle', which corresponds to the size of image on the retina. Visual angle may be calculated by finding the arctan of the size of the object viewed divided by the distance from the object. For example, the visual angle subtended by a letter (0.5 cm approx) on an

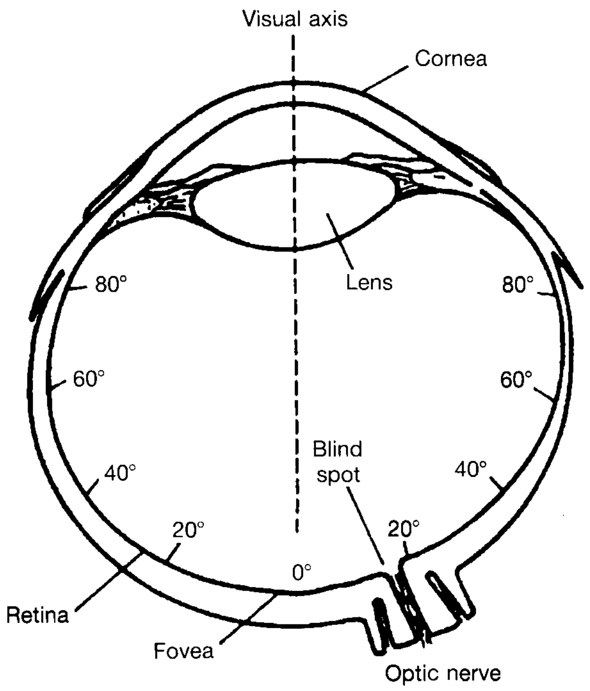


Fig. 2 Cross section through the eye showing the passage of light along the visual axis, with degrees of visual angle from the fovea.

ordinary 80 × 25 column computer display at arms length (70 cm approx) would be: Arctan $(0.5/70) = \arctan(0.00714) = 0.41°$, or 24′ of arc of visual angle ($1° = 60$ minutes).

Another important feature of the eye is the presence of the blind spot. This is the point in the retina where no light receptors are present but where information from the retinal receptors is routed to the brain. This illustrates the tremendous data compression taking place within the visual system as signals from the 125 million receptors are coded and sent out through the one million nerve fibres of the optic nerve. The coding and decoding of signals from the eye is still being intensely researched, more details of which may be found in Coren, Porac and Ward (1984), Savoy (1987) and Livingstone (1988).

2.2.1 The retinal receptors The rods allow vision at low levels of luminance down to hundredths of a candela per square metre, (cd/m^2). This type of vision is called 'scotopic' vision, and is sensitive to movement and light intensities but not to colour. The cones function at high levels of luminance and give colour or 'photopic' vision at levels of luminance of a few cd/m^2 or more. As luminance levels rise to between 1 and a couple of cd/m^2 there

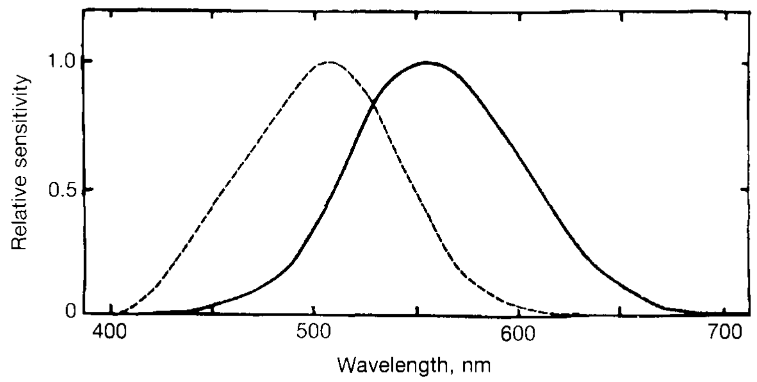


Fig. 3 The sensitivity of the eye under scotopic vision (broken line) and photopic vision (solid line) conditions by wavelength.

comes a point where both rods and cones are active and this is the 'mesopic' form of vision. In normal daylight illumination levels the rods have little or no contribution to make to what we see, (Hunt, 1987; Long and Garvey, 1988).

Rods and cones are not equally sensitive to all wavelengths of light, with rods (scotopic vision) being markedly more sensitive to the lower wavelengths (blues, violets) compared to photopic vision, which is more sensitive to the longer wavelengths (reds). Figure 3 shows the different sensitivities of the two systems. This difference means that the brightness of a blue and a red stimulus compared under normal levels of illumination will change when they are viewed under dim conditions, with the blue stimulus appearing relatively brighter and the red relatively darker. Note that the scale on the ordinate of Figure 3 is a scale of relative sensitivity to wavelength and the rods are actually many times more sensitive to each wavelength than the cones. Also, for colours viewed in normal daylight conditions the eye will be most sensitive to the yellow-green part of the spectrum and least sensitive to blues and reds. In other words, even though the actual physical luminance might be the same, light of wavelength 560 nm (a yellow-green) will be sensed as approximately twice as bright as a light of wavelength 510 nm (a green). This effect may be demonstrated by viewing a prism spectrum or a rainbow where the yellow-green areas of the spectrum appear much brighter than the red and blue parts.

2.2.2 Cones There are three different types of cone, and these are differentially sensitive to wavelengths of light, see Fig. 4. These cones are frequently called 'blue', 'green' and 'red' but this can cause confusion and are better described as short, medium and long wavelength sensitive cones respectively. For example, the so-called red cone has a maximum sensitivity in the yellow-orange part of the spectrum, and is also sensitive, although less strongly, to many of the wavelengths also covered by the green cone. The

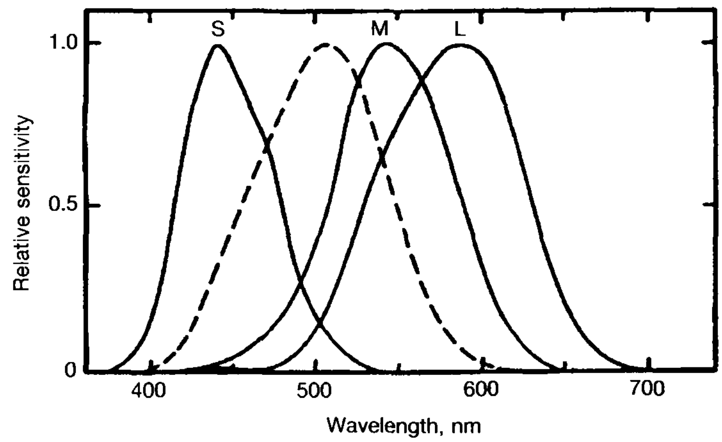


Fig. 4 Spectral sensitivity of the three types of cone, with the broken line showing scotopic (rod) vision. Where *S* = short, *M* = medium and *L* = long wavelength sensitive cones.

different types of cones are not spread evenly across the retina, the central region of the fovea being blue-blind, red and green sensitivity being limited to about 30° from the fovea, (see fig.2) but with blues and yellows still be able to be sensed at up to 60°, (Coren et al, 1984). Research has also shown that there are different amounts of the three types of cone within the eye: estimates usually approximate to 40R:20G:1B. This has a number of effects on the sensation of colour, not least that the cones have to be stimulated in this proportion in order to experience the sensation of white.

The colour of an object is therefore determined by the ratio of cone responses. A colour with a hue of 550 nm will evoke a large 'red' response, a weaker 'green' response, and no response from the 'blue' cones.

2.3 Colour defective vision

It is well known that there are people who cannot discriminate colours as well as others and such people are commonly labelled 'colour blind'. True colour blindness is however very rare, and perhaps a more accurate label for this problem would be 'colour defective'. There are various types of colour defective vision, the causes of which are still not perfectly understood but may be classed into three categories: those in which either all of the rods or all of the cones are missing, those in which one of the different types of cone is missing, and those where there is a reduction in the sensitivity of one (or more) of the cone types in increasing order of occurrence. People who have normal colour vision are called 'trichromats', referring to the three normally functioning sets of cones, with 92% of males and 99·5% of females being normal trichromats, (Hunt, 1987; Wyszecki and Stiles, 1982).

People who suffer from true colour blindness, called 'cone monochromatism' experience many visual problems, not least of which is the inability to see objects at the fovea, resulting in functional 'day blindness'. Those who have no rods, but working cones, are functionally blind at low levels of illumination.

By far the most common form of colour defective vision occurs within approximately 4·9% of males and is called deuteranomaly. This is a result of the green cones having a spectral sensitivity shifted along the wavelength axis towards the normal short wavelength (red) cones, (see fig.4), with a resulting loss of discriminations based on the ratio between red and green cones. The reason why colour defective vision is much more widespread in males than in females is that the genetic information is carried on the sex chromosomes. People with deuteranomalous vision can still see greens but will tend to confuse certain reds and greens, and some reds and greens with grey, (Long 1984). It is usually possible to predict colours which will be confused by people with such abnormalities, (Wyszecki and Stiles, 1982; Meyer and Greenberg, 1988). All of the defective colour vision problems will be successfully picked up by the familiar Ishihara colour vision test. However the Ishihara tests are not sensitive enough to say exactly which colour defect is present, e.g. lack of sensitivity in, or lack of, a set of cones, and usually more sophisticated and complex tests such as the City University or Farnsworth-Munsell tests are needed to give an exact diagnosis, (Fletcher and Voke, 1985; Meyer and Greenberg, 1988).

2.4 Models of colour vision

Thomas Young and later Von Helmholz were the first to realise, and then show with colour mixing experiments, that people with normal colour vision only need three suitably chosen primary colours to be able to match any colour stimulus. This gave rise to the theory that people did not need retinal cells individually sensitive to each wavelength of visible light, but only cells sensitive to short (e.g. blue), medium (e.g. green) and long (e.g. red) spectral wavelengths. Evidence from colour defective observers, whose deficiencies seemed to result from a lack or insensitivity in one or more of these cells, also gave weight to this 'trichromatic theory'. Actual physiological evidence for these cells (which are of course the cones), has only been found in the last 25 years as microelectrode techniques improved, (Brown and Wald, 1964; Bowmaker and Dartnell, 1980).

The main problem with trichromatic theory is that when people are asked to describe colours, or to choose 'pure' colours from a set they tend to use four hue names, not three. These unique colours, which occur in cultures other than the West, are red, green, blue and yellow, (Zollinger, 1988; Uchikawa and Boynton, 1987). Ewald Hering used such evidence to fashion his 'opponent theory' of colour vision. In this theory the four primaries are in two sets of opponent pairs: yellow-blue and red-green, with a third primary pair black-white. Figure 5 shows the inputs to the opponent pairs from the cones, and the resulting information. Note that these opponent properties

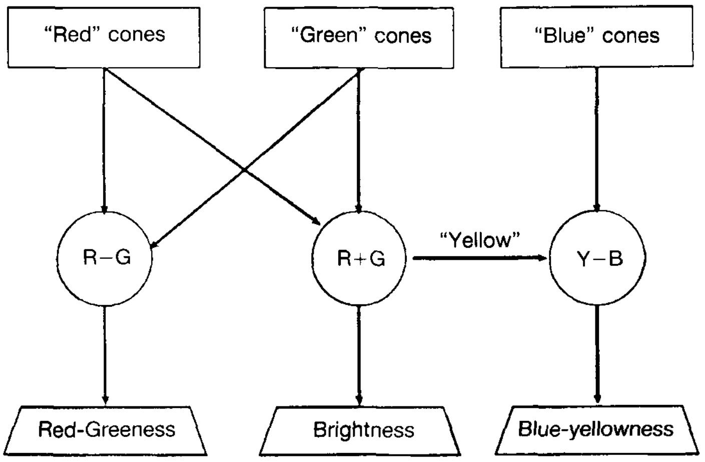


Fig. 5 Schematic diagram of the cone inputs to the opponent colour process in vision.

mean that whilst we can easily see yellow-reds or greenish blues, we cannot see greenish reds or yellowish blues. In 1958 Svaetichin and MacNichol showed how some cells in the retina do indeed have the property of taking information inputs from the cones, and then responding, in the example of the blue-yellow opponent cell, with a low voltage to blue and a high voltage to yellow. Since this time further evidence has been found which shows that the story becomes even more complex as the nerve carrying information is traced to the brain, with neurones which respond specifically to, for example, green horizontal lines being found (Michael, 1981). For an up-to-date review of the research and the current theories of colour vision see Boynton (1988).

3 Colour as a physical phenomenon

3.1 Hue, saturation and brightness

The spectrum is an inadequate way of describing colour for many reasons, principally because a spectrum contains information only about 'hue'. If people are asked to place coloured tiles or lights in an ordered sequence it is found that many colours which have the same hue still differ in some respects. For example some colours may appear washed out, almost white, but still have a discriminable hue. When the light from these colours is analyzed it becomes apparent that this difference is associated with the 'purity' of the colour, or the extent to which a hue has been diluted by being mixed with other hues. When a hue is mixed with all other hues the effect is to make the colour more like white light (which Newton showed is the sum of all colours). This purity is called 'saturation'; for example a pink is an unsaturated red.

A 'colour circle' may be formed by thinking of the centre as white, and the

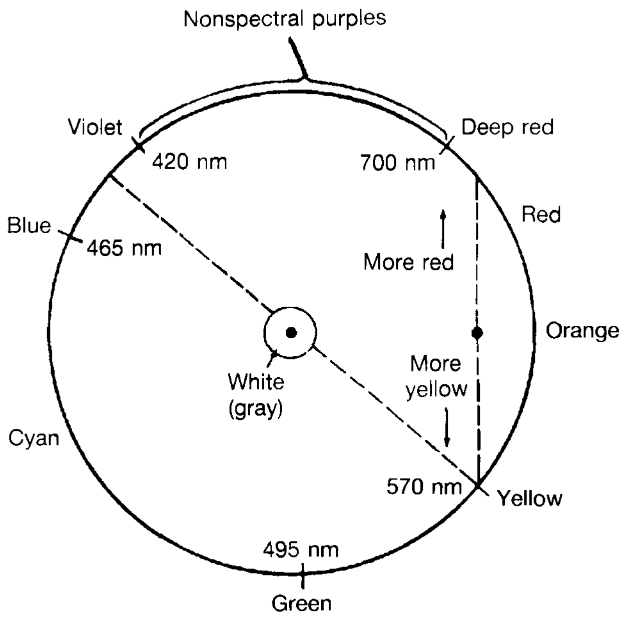


Fig. 6 The colour circle, illustrating the result of mixing a yellow and a red to get an orange colour.

outer edge as containing the fully saturated spectral hues (and non-spectral purples), as in Fig. 6. Hue angle is often referred to in the literature and is the angle between a reference hue (e.g. a fully saturated red on a monitor) and other colours around the hue circle. Colours on opposite sides of the circle are called 'complementary colours' and have a number of interesting features which will be described later.

If two colours are matched on hue and saturation but still appear to be different, this difference will be due to the respective lightnesses of those colours. Whilst hue and saturation can be represented by a two dimensional diagram, the introduction of the third (and final) dimension needs to be represented by a 'colour solid'. Figure 7 shows a representation of such a colour solid. Brightness is a psychological correlate of luminance, which whilst easy to understand is difficult to incorporate into a colour space because hue, saturation, the relative lightness of an object and environmental factors such as illumination will effect how bright such an object will appear. Consequently colour spaces often employ hue, saturation and lightness (which ranges from black to white), or other combinations to avoid confusions with bright to dark scales.

3.2 Additive vs. subtractive colour mixtures

Only very rarely will people see colours which are truly 'monochromatic' outside of experimental labs. Monochromatic, meaning one-colour, refers to

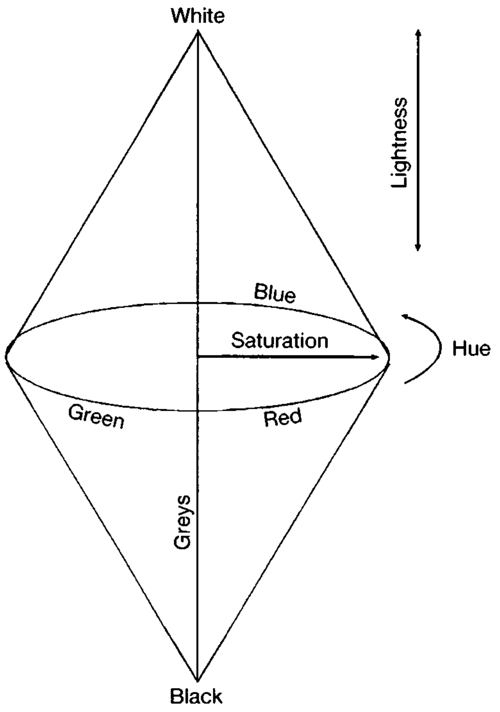


Fig. 7 The hue, lightness and saturation colour solid, showing colour descriptions of various points of the space.

a colour which consists of light of only one wavelength. Most colours we see in the real world are mixtures of many wavelengths, and the overriding hue sensation of each mixture is called its 'dominant wavelength'. When two colours are mixed, e.g. two coloured lights, we will see a new colour with a combination of the properties of its constituent parts. When these colours are mixed in certain ways the eye cannot discriminate the individual constituents, but instead will sense a single colour. Two colours which appear to be identical can consist of different mixtures of colours, in which case the mixtures are called 'metamers'.

There are two types of colour mixture. The first is used to calculate the effect of mixing paint or pigments. Most people have dabbled with poster paints at school and know that blue added to yellow makes green, and a number of different paints makes a grey, and eventually a black. This form of mixing takes away light energy from the stimulus and so is called subtractive mixing, see Fig. 8. The type of mixing required when viewing luminous objects such as computer displays is called additive colour mixing because with each colour added light energy is given to the stimulus, with many colours together forming light greys, and eventually a white. Additively mixing yellow and blue together does not give rise to a green, but to a grey. A highly

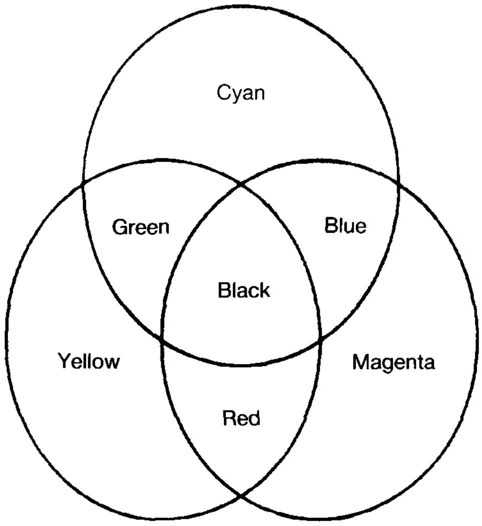


Fig. 8 Subtractive colour mixture using three patches of paint.

saturated green cannot be created by additively mixing lights, but requires a pure, primary colour to be realised. One of the effects of additive colour mixing that people find most surprising is that by mixing red and green light yellow is produced (which would produce a grey in subtractive mixing), see Fig. 9. The difference between additive and subtractive mixing is one strong

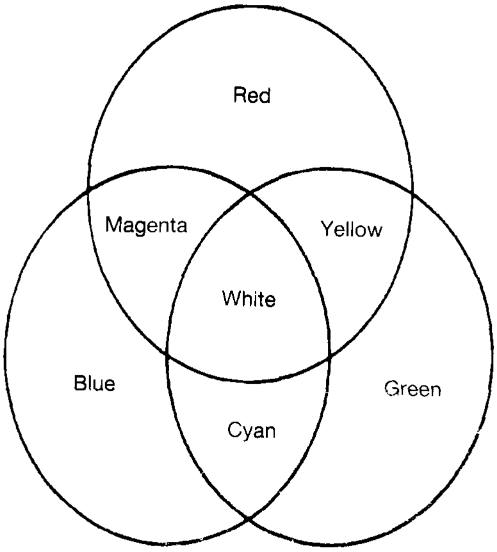


Fig. 9 Additive colour mixture using three beams of light.

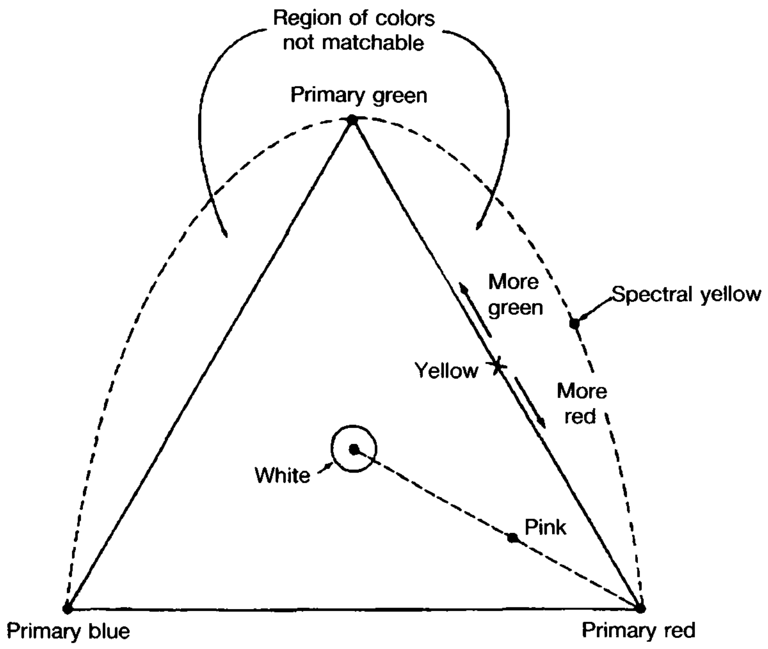


Fig. 10 Specifying colours using a colour triangle.

reason why designers have to be very careful when interpreting the results of experiments based on pigments (such as coloured papers) and applying them to CRT displays.

One of the more important properties of the colour circle is the ability to calculate the result of additively mixing colours. If a line is drawn from a red to a yellow in the colour circle, the colour mixture seen by adding equal proportions will be equal to the centre of that line, which will be an orange colour, (see fig.6). The orange created will be less saturated than either the red or the yellow, which is demonstrated by the centre of the line lying further away from the perimeter than the two end points. It will always be the case when additively mixing colours that no mixture of colours can ever be as pure (saturated) as a monochromatic or spectral colour because of the convexity of the circle, (see Fig. 10). If three colours are used in equal propertions to form a mixture, then the resulting colour will be at the centre of the triangle made by these colours within the colour circle.

3.3 Colour circles to colour spaces

By selecting three widely spaced spectral hues it is possible to reproduce most hues by adjusting the relative amounts of the three colour sources. Geometrically, the triangle (or 'gamut') formed between these hues covers most of the colour circle. These colours are called 'primaries' and have the property that the hue of any one of them cannot be replicated by mixtures of the other two. Traditionally the red, green and blue hues which are used to calculate mixtures have wavelengths of 700 nm, 546·1 nm and 435·8 nm respectively. A three dimensional colour space may be constructed by plotting proportions of the red, green and blue values on its three axes. This may be reduced to two dimensions given that the three proportions must sum to 1.

However, as mentioned above, because of the curvature of the colour circle it is impossible for all hues to lie within the gamut formed by any three primaries. This is illustrated in Fig. 10 which shows the curved area of unrealisable colours. This shortcoming may be overcome by permitting a negative proportion of one primary. In practice this works by adding a primary to a test colour and then matching the outcome, for example:

$$\text{Test colour} + W(r) = W(g) + W(b)$$

$$\text{i.e.}\quad \text{Test colour} = W(g) + W(b) - W(r)$$

where $W(r)$, $W(g)$ and $W(b)$ are different amounts of red, green and blue. The necessity of using negative weights is not thought to be conceptually desirable and in 1931 the Commission International de l'Éclairage (CIE) proposed a solution based upon imaginary (or super) primaries which themselves lie outside the colour circle but whose gamut encloses all hues that can be sensed. The super primaries were termed X, Y and Z, (Hunt, 1987; Wyszecki and Stiles, 1982).

The coefficients X, Y and Z were selected so the amounts needed to match any other colour in the circle would be positive. The relative magnitudes of these primaries or 'tristimulus values', calculated by $x = X/(X+Y+Z)$ etc., are called chromaticity coordinates and are used to plot colours on the 1931 chromaticity diagram. This is shown in Fig. 11 which illustrates how all of the colours may be reproduced using only positive combinations of the super primaries. Note that, because $x+y+z=1$, if any two of the coordinates are known the third can be calculated by taking the difference of the sum of the two known coordinates from 1. Therefore only a two dimensional diagram is needed to represent x, y and z, and by tradition x is on the abscissa and y on the ordinate. See appendix B for instructions on how to derive chromaticity coordinates.

It was mentioned in Section 3.1 that the spectrum did not distinguish between colours of differing lightnesses. As only the relative values of the super primaries are used to identify the hue of a colour, the CIE defined the absolute value of Y to represent the luminance (which is a precise physical measure correlated with perceived brightness) of the colour as based upon colour matching experiments. Thus a colour may be uniquely identified by the triple 'x,y,Y', (pronounced little-x, little-y and big-y).

3.4 Perceptual spaces, Munsell, UCS, etc

The CIE 1931 diagram is very useful as a means of predicting the results of colour mixing and the physical relationship between colours. But the

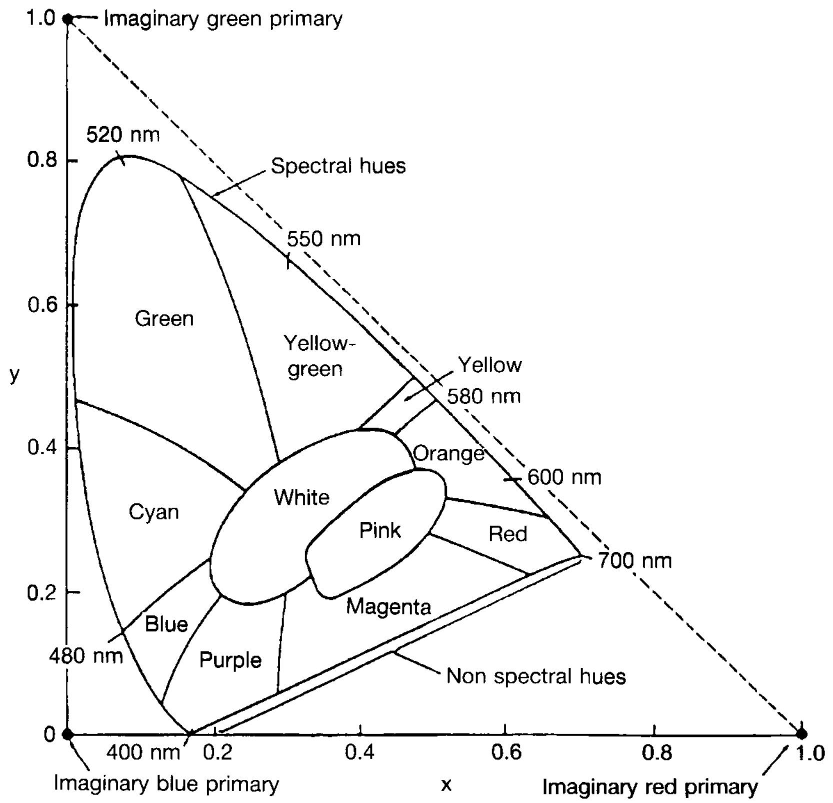


Fig. 11 The CIE 1931 *xy* chromaticity space showing the three imaginary primaries, and colour names approximately associated with regions of the space.

physical production of a colour and its sensation or perception by a person are very different processes. If colours were generated from the (x,y) diagram by taking small, equal steps, the colours would not be sensed as being equally spaced. In some instances there would be little perceptual difference between steps, and in others quite big differences.

In 1976 the CIE developed the Uniform Chromaticity Scale diagram or UCS diagram in which much of the perceptual non-uniformity of the (x,y) diagram was removed. The UCS or u', v' (called u-dash, v-dash) diagram is a simple transformation of the 1931 diagram, and whereas in the 1931 space the ratio between the size of the largest just noticeable difference (JND) to the smallest JND between colours was 20:1, in the 1976 diagram this had been reduced to 4:1. The 1976 u',v' diagram is shown in Fig. 12.

Chromaticity diagrams such as the x,y and the u',v' diagrams are of restricted use to the display designer because they only refer to the hue and saturation of colours which have the same luminance (i.e. are 'isoluminant'). Instead for

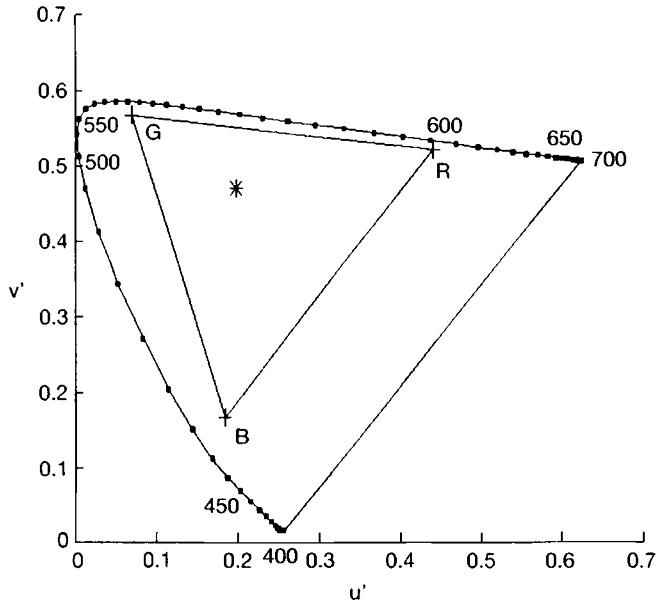


Fig. 12 The CIE CIE *u'*, *v'* diagram showing the gamut of colours which would be enclosed within the region of typical monitor phosphors, where '*' is the white point.

specifying colours exactly on colour CRT displays the CIE 1976 $L^*u^*v^*$ (pronounced L-star, etc.) colour space should be used. The lightness measure L^* transforms luminance readings into a scale with white at $L^* = 100$ and black at $L^* = 0$, the greys are given by L^* values in between white and black. L^* must be based on a reference white, the white produced by 100% of the RGB guns is often used (Carter and Carter, 1983). u^* and v^* are quantities based on the u' and v' coordinates of a colour and a reference white.

It is possible to give a good correlation of the perceptual difference between two colours by measuring the $L^*u^*v^*$ of two colours and then using the CIE's 1976 colour difference formula (called Delta E or ΔE^*_{uv}). Most of the formulae for the transformations mentioned above are given in appendix B.

3.4.1 The Munsell and other colour order systems One of the most popular colour order systems (a method for dividing up colours depending on certain properties the space should contain) is the Munsell system. This system was first introduced by the artist Munsell in 1905, and has since been widely updated and refined, (Nickerson, 1976). This system uses the three dimensions of hue, value (similar to lightness) and chroma (similar to saturation) to define colours. Within this space the perceptual difference between any two HVC combinations should be the same. This system is widely used by people who need to describe surface colours, such as artists and dye manufacturers, and samples of the space are usually produced in the form of books of coloured tiles, called the 'Munsell book of colour'. The tiles are arranged such that each page represents a vertical slice of hue, with tile lightness arranged vertically, and chroma horizontally for each hue. A restricted set of colours is defined within the Munsell space. There are 40 hue pages with 10 value (lightness) steps, the number of chroma steps being dependent on the HVC combination being viewed, as some colours are naturally more highly saturated than others.

Other well known colour spaces include the Ostwald space, the Natural Colour Order System (NCS) and the Optical Society of American (OSA). The Ostwald colour system is similar in shape to the space shown in Figure 7 and is based on principles of additive colour mixture. The NCS is a perceptually based colour appearance system based on many measurements of coloured papers, and derived from the work and theories of Hering (see section 2.4). The OSA system is a space in which equal distances very closely relate to perceived differences, but which consequently lacks dimensions such as hue or saturation. See Hunt (1987) and Wyszecki and Stiles (1982) for examples of other colour order systems.

It should be noted that many people have cast doubt on the application of the CIE measures to CRT displays without qualification. Finding perceptually different colours for such displays is also problematic, and is probably dependent on the task to be performed, (Van Laar and Flavell, 1988; De Corte, 1986; Mollon and Cavonius, 1985).

4 How do visual displays produce colour?

4.1 CRT displays

The most common means of displaying coloured information is by utilising colour cathode-ray tubes (CRTs). In a CRT colours are produced by 'shooting' streams of electrons within a vacuum tube at colour phosphors embedded in the inside surface of the monitor or television screen. When stimulated in this way the phosphors emit light. Groups of red, blue and green phosphors are distributed evenly over the screen and respond to one of three corresponding electron guns. The different colours are produced by additively combining the light of the three phosphors, and when seen from a distance the colours are perceived as one (see section 3.2). A maximally saturated colour such as red is produced by stimulating the red phosphor at maximum. When maximum red and green phosphors are stimulated together the maximum saturated yellow is produced, blue and red give magenta and blue and green phosphors together produce cyan. All three phosphors stimulated together produce white, and the absence of all three is seen as black. Because phosphors in older displays were only capable of being stimulated at two levels, on and off, only eight colours were available (i.e. 2^3 colours). This may be contrasted with the huge number of colours produced on more modern displays by varying the intensity of the energy falling on each phosphor. For example, if the voltages applied to the electron guns are able to be 'stepped over' 256 different levels then 256^3 different combinations are possible, giving in theory over 16 million different colours. A common misunderstanding is to believe that people can discriminate all 16 million colours produced by such a display; however the actual estimated number of hue, saturation, lightness combinations that are just perceivably different (see JND in the glossary, appendix A) on a display is closer to 1.2 million (Tajima, 1983) under very good viewing conditions.

When stimulated by an electron beam the phosphors on the screen glow for a short time, allowing time for a raster-scan system to operate. With this system the electron gun stimulates each group of phosphors in turn, usually each group on a horizontal line, and then through each vertical line of the screen in turn. The phosphors are re-stimulated or 'refreshed' at a rate of many times (typically 50) a second, which is usually fast enough to give an illusion of a static colour screen display.

One of the problems with using this form of screen display technology is that sometimes, with many phosphors to refresh (high resolution), wide angled displays and bright colours, the screen may be seen to flicker, especially in the periphery of vision, (Tyler, 1985). In many cases flicker may be reduced by decreasing the brightness of the CRT, or by running the CRT at high refresh rates e.g. 100 times per second or higher. (Tyler, 1986).

Many algorithms exist for the display designer to predict the perceived colour of a display, given various phosphor types, filters and viewing environments, in particular Viveash and Laycock (1983) and Hunt (1987).

4.2 LCD and other colour display technologies

There are three main ways of producing colour Liquid Crystal Displays (LCDs): nematic crystal, smectic liquid crystals and ferroelectric crystals. These must usually be combined with backlights, polariser or filters. Liquid crystals have the physical properties of liquids but the optical properties associated more with solids. When a current is applied across a liquid crystal reservoir the crystals change from a random state to an aligned state, which if the reservoir is correctly positioned can change the viewed stimulus from white to black, or from one colour to another. LCDs are light scattering rather than light emitting, and although characters and graphics have good readability in high illumination, in medium or low illumination conditions the display must usually be backlit in order for the characters to be seen.

Although problems with current colour LCDs include poor resolution, poor contrast, bad response times and small colour gamut, because of the low power demands, light weight, small physical size and robustness to violent movements, these displays are still being actively researched, especially in the avionic and military industries.

Other types of technology include thin film transistor (TFT) displays, light emitting diode (LED) displays and gas plasma displays. These all suffer from many disadvantages compared with the highly developed understanding of the CRT and it is very unlikely that any will take over from CRTs in general computer and other information displays except in specialist uses such as portable computers.

Given the ubiquitous nature of the CRT and the lack of data based on other forms of screen display, this paper has concentrated on the use of colour in CRT displays.

4.3 Colour spaces based on display technology

Colour spaces based on physical and perceptual criteria such as the CIE, Munsell and Ostwald spaces *must* be used in cases where the reproduction or cross referencing of colours between displays is to take place. The alternative is to use colours based on the RGB values of the monitor's CRT guns, where colours are specified in percentages of each gun so that the most saturated yellow colour would be R = 100%, G = 100%, B = 0%, etc. It is crucial to understand the distinction between the colour primaries dicussed above and the RGB gun values. These latter are dependent upon the phosphors used in the construction of the CRT and the energy being transmitted by the guns to excite them. The RGB colour space is shaped like a cube, implying the independency of the three guns, unlike x,y,Y which demonstrates the physical relationships between the dimensions. The derivation of colours in RGB space cannot be easily understood (Schwartz, Cowan and Beatty, 1987; Murch, 1984) and more importantly, colour produced by a particular combination of RGB values on one monitor are very unlikely to produce the same colour on another monitor. The lack of correspondence between monitors may be due to different ages of the monitor, the use of different phosphors, filters or driving voltages, or even the direction in which the monitor is facing (in or out of the earth's magnetic field). It is possible to overcome these problems by calibrating a monitor so that a software program interprets instructions given in (e.g.) Munsell values and outputs the appropriate RGB values to the screen, (Cowan, 1983; Glassner, 1989). However, such methods of calibration take a long time, need frequent updates for most monitors and require specialist colour measurement equipment, at least to initiate the calibration.

The RGB space produces colours which must be combined in additive ways in order to produce the desired colours. As mentioned above, it is possible to work with additive colour mixtures but people do not find them intuitively useful and perform less well with them than with other more perceptually-orientated spaces, although they can be useful in specific tasks. One such space is the HLS space, based on the Ostwald colour system with a double hexacone shape; Fig.7 shows a similar double cone shape. Brown and Cunningham (1989) describe a number of such systems and their advantages and disadvantages. A simple algorithm exists to convert from RGB to HLS values and back (see Foley and Van Dam (1982), pp.618–619) but it must

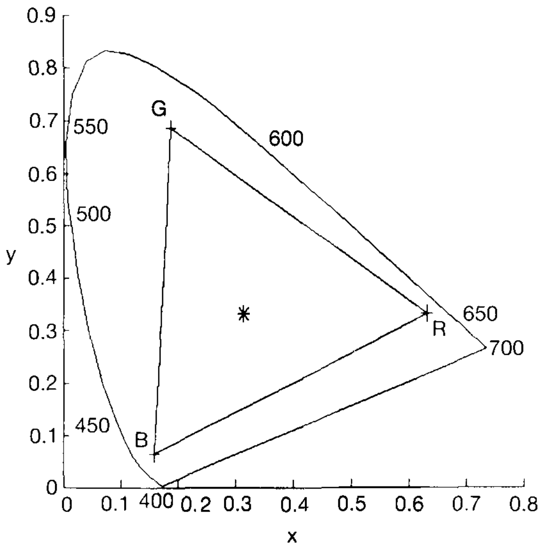


Fig. 13 CIE 1931 *xy* chromaticity diagram showing the gamut of colours enclosed within typical monitor phosphors, where '*' is the white point.

always be remembered that within this space H (measured in degrees), S(%) and L (%) are only approximations to the hue, saturation and lightness of the physical and objective spaces described in section 3.1.

As previously mentioned in section 3.2 if there are two CRT phosphors which are plotted in an appropriate colour space, such as G and B in Fig. 13, then all the colours which lie along that line may be produced by different mixtures of the phosphors. In the same way if there are three phosphors, then all the colours lying on and within the triangle formed in the colour space are realisable by mixing these colours. One important point to remember is that the colours realisable on a CRT give only a restricted part, or 'gamut' of all colours that people can see; again this may be seen in Fig.13.

5 How do people perceive colour?

When one colour is seen on a background of another the foreground colour seems to change hue in a direction roughly towards the complementary colour of the background. In the same way bright backgrounds tend to make the target colour appear darker, while dark backgrounds usually make the target appear brighter than is actually the case. These effects occur without any change in the physical light coming into the eye from the target. In other words there is often a difference between the physical colour of the target which is sensed, and the colour that is actually seen, or perceived, (Van Laar and Flavell, 1988).

It is very important when creating colour displays that the designer is aware of the effect of background and nearby colours on the perceived colour of a stimulus. This is especially important when absolute colour judgements are to be made, such as when colour coded brain scans or temperature analyses are to be used, or in other situations where colour codes are used to portray meaning.

5.1 Chromatic adaptation and colour contrast

When coloured objects or objects illuminated by light of a certain colour are viewed, after a time the sensitivity of the eye to that colour changes; this is known as chromatic adaptation. For example, viewing a display with a blue background or illuminated by blue light will tend to fatigue the blue part of the visual system, with the result that other objects will appear to be yellower than they are normally (as yellow light is complementary to blue). Note that yellower does not mean yellow, for example a blue green on a blue background will actually look a purer green as the yellow light is perceived as mixed with the blue light. However this effect of coloured backgrounds does not usually become a problem in colour displays as the viewer will normally not stare at the screen for long enough at one time, but will swap his or her gaze to paper, to the keyboard, etc. Differently coloured illuminants will cause more of a problem with colours seen under tungsten light and daylight looking quite different. If one is required to look at one part of a colour display for a long period of time, such as when waiting for a value to appear at a certain cell of a spreadsheet, it is possible that this fatiguing effect may occur across only one part of the retina, the result being an after-image: the colour of the after-image again being complementary to that of the 'inducing' image.

Similar results are observed with 'simultaneous colour contrast' effects, where colour patches on a screen or other stimulus will induce a complementary colour in another part of the screen, and this hue shift will affect how objects are seen, (Krauskopf, Zaidi and Mandler, 1986). This effect is instantaneous and due to an automatic neurological response based on the spatial arrangement of the display and not due to fatigue. Effects can be very strong on some displays. When absolute colour judgements have to be made, such as when identifying areas on a colour map, these effects can cause large changes in the colours perceived. One of the current projects at Imperial is concerned with developing algorithms which will take such effects into account and change the displayed colours automatically.

The contrast effects discussed above may be termed 'induction' effects because a colour difference is induced onto a displayed object. The opposite effects also occur, where the colour of an object affects that of the background, and these are called 'assimilation' effects. Which effect actually occurs is not yet fully understood but evidence from brightness assimilation experiments has shown that whether assimilation or contrast occurs depends on whether the displayed object is seen as being the figure (i.e. in the foreground) or the ground (i.e. the background). For example, a display may be seen as a yellow object on a blue background, or a blue object with a hole showing through to a yellow background, (Festinger, Coren & Rivers, 1970).

5.2 *Other visual effects on colour perception*

5.2.1 Effects on colour discrimination The colour of very small objects ($<$ 10–15 minutes of visual angle, see section 2.2) is usually perceived as different from that of larger objects of the same colour. One observation is that very small objects are seen as much less saturated. Another effect is that in the discrimination of small blue objects the visual system acts as if no blue cones were present. This effect is called 'small field tritanopia' after the colour vision deficiency with the same symptoms. Blue stimuli are also problematic because of the very small input to the luminance channel. The lack of such information from blue stimuli to the edge detecting mechanisms which depend on luminance differences means that letters on a screen which differ from a background only in blue will be very difficult to read, (Beck, Sutter and Ivry, 1987; Savoy, 1987). However in most instances blue lettering on displays is still satisfactorily seen because of the hue and saturation contrast between the letters and the backgrounds.

As mentioned in section 2.2 colour perception in the periphery of vision (away from the fovea) is reduced, the perceived effect being a desaturation of the colours in the periphery. This will of course affect colour discrimination on the larger displays currently becoming popular (e.g. colour Sun, Mac II machines). Information codes employed at the edges of such displays should therefore be blues or yellows, (because of the way the different cones are distributed, see section 2.2.2), be highly saturated and should have a large brightness contrast compared to other codes and colours if they expect to be seen.

As colours increase in brightness they appear to change hue, with violets and blue-greens (wavelengths <510 nm) becoming bluer and yellow-greens and reds (wavelengths >510 nm) becoming yellower. This shift in hue, called the Bezold-Brücke effect, is also seen when the colour of an object on a display is made to look brighter by surrounding it with a darker background.

5.2.2 The McCollough effect McCollough found that when people viewed coloured stripes in particular orientations, and then viewed black and white stripes in similar orientations, they would see imaginary, or subjective colours in the black and white grid. Even more surprising, after just 10 minutes or so of viewing green and red stripes these 'colour contingent after effects' lasted sometimes for months. These illusions are not therefore due to retinal fatigue but seem to have a neuronal or cortical component. This effect seems to be specific to the orientation of the lines involved and does not seem to work for the blue-yellow opponent system, (Savoy, 1987). Some computer operators have reported that after long term viewing of displays with green text on black backgrounds, then viewing black and white displays made characters and letters look pink, (Greenwald, Greenwald, and Blake, 1983) and there is even evidence that the effect occurs only with understandable text, and not with nonsense words (Allen, Siegal, Collins and MacQueen, 1989).

5.2.3 Fluttering hearts phenomenon When two highly saturated complementary colours are combined on a display a disturbing visual effect occurs, where the colours appear to flicker or shimmer, called the 'fluttering hearts phenomenon'. This effect, so called because of the shape of the stimuli in the first experiments, occurs mainly at the boundaries between coloured objects and is caused by differences in the adaptation of the cones at these edges as the tiny, constant, movements of the eye move the edges of the object across the retina. These effects may cause problems in colour displays but can be negated by using colours which are less different in hue and which are less saturated.

5.2.4 Chromatic abberation and colour stereopsis Coloured light entering through the edges of the lens of the eye is not focused on the retina in the same plane as light entering through the centre of the lens. This produces 'chromatic abberation'. Because blue light is focused by the eye at a nearer point than green light, in turn focused nearer than red light, the eye, in order to maintain a sharp retinal image might need to change focus between objects. The result of such changes can be that the muscles which focus the lens (a process called accommodation) become tired or fatigued. Another effect would be that if text in one colour on a display were in focus, text of a colour far away in spectral composition would not and might appear to be in a different plane.

However, because of the lack of blue receptors in the fovea, the eye only needs to change focus in the green to red parts of the spectrum. Colours found on most VDUs are not monochromatic but are mixtures of hues from different parts of the spectrum, with the colour's dominant wavelength determining the perceived hue. Also the eye's focus to a visual display is determined by factors other than stimulus colour, such as eye convergence and stereo information from the visual system (Walraven, 1984). According to Campbell & Durden (1982), people who do not have marked lens ageing will rarely find, under normal viewing conditions, that chromatic abberation causes focus problems or fatigue.

Colour stereoscopic effects are caused by a combination of chromatic abberation and eyes aligned so that in most cases reds are seen as being in front of blues. However some individuals see blues in front of reds, and rarer still some see both reds and blues as lying on the same plane, (Hunt, 1987).

Colour stereoscopy is most pronounced for highly saturated combinations of red and blue on black or dark backgrounds (Hunt, 1987), which are common colour coding formats for some types of computer displays. Given that the screen designer would not want to harness this visual effect to show say, red objects as standing out from a blue background, (not recommended because of other visual effects such as fluttering hearts phenomenon) colour stereoscopy may be overcome by choosing brighter backgrounds, less saturated colours and colours with less difference in spectral hue.

5.2.5 Age and physical condition The ability to discriminate colours changes systematically as people age, the lens becoming progressively yellower so that all light must pass through a gradually darkening yellow filter (Coren and Girgus, 1972). There is also a deterioration of blue-yellow discriminations as people get older (Verriest, 1972), and a reduction in the optical transmittance of the fluids in the eye so that colours are perceived as darker and less saturated (Murch, 1984). Unlike the colour deficiencies discussed in section 2.3, which are congenital, some diseases result in loss of colour vision or discrimination. Diabetics suffer losses in their blue-yellow discrimination abilities, and alcoholics are prone to reductions in blue-sensitivity (Coren et al 1984).

6 Summary

To understand how to employ colours effectively on displays it is necessary to appreciate the physics of colour science, the physiology of colour vision and the perceptual factors likely to be encountered when using colour displays. This paper has examined all of these topics and has given the reader an appreciation of the problems involved. For further information, consult the references given below and in the text. Note that explicit advice on how to colour code displays based on the information given in the present paper will be contained in the next paper of the series.

References

ALLEN, L.G., SIEGAL, S., COLLINS, J.C. and MacQUEEN, G.M., *Color aftereffect contingent on text.* Perception and psychophysics, 46(2), 105–113, 1989.

BECK, J., SUTTER, A. and IVRY, R., *Spatial frequency channels and perceptual grouping in texture segregation.* Computer vision, graphics and image processing, 37, 299-325, 1987.

BOWMAKER, J.K. and DARTNELL, H.M.A., *Visual pigments of rods and cones in a human retina.* J of Physiology, 298, 501–11, 1980.

BOYNTON, R.M. *Color Vision.* Ann. Rev. Psychol, 39, 69–100, 1988.

BROWN, J.R. and CUNNINGHAM, S., *Programming the user Interface: Principles and examples.* John Wiley, London, 1989.

BROWN, P.K. and WALD, G., *Visual pigments in single rods and cones of the human retina.* Science, 144, 45–52, 1964.

CAMPBELL, F.W. and DURDEN, K., *The visual display terminal issue; a consideration of its physiological, psychological and clinical background.* Ophthal. Physiol. Op. 3, 175–92, 1982.

CARTER, C.C. and CARTER, E.C., *CIE L*u*v* Color-difference equations for self-luminous displays.* Color Res. and Appl., 8(4), 252–3, 1983.

Commission Internationale de l'Éclairage proceedings. Cambridge University Press, Cambridge, 1931.

COREN, S. and GIRGUS, J.S., *Density of human lens pigmentation: in vivo measures over an extended age range.* Vision Research, 12, 343–6, 1972.

COREN, S., PORAC, C. and WARD, L.M., *Sensation and perception,* 2nd Ed., Academic Press, London, 1984.

COWAN, W.B., *An inexpensive scheme for calibration of a colour monitor in terms of CIE standard coordinates.* Computer Graphics, 17(3), 315–21, 1983.

De CORTE, W., *Finding appropriate colours for colour displays.* Col. Res. and Appl. 11(1), 56–61, 1986.

DURRETT, H.J., *Color and the Computer.* Academic Press, 1987.

FLETCHER, R. and VOKE, J., *Defective colour vision.* Hilger, Bristol, 1985.

FOLEY, J.D. and VAN DAM, A. *Interactive computer graphics.* Addison-Wesley, 1982.

GLASSNER, A.S., *How to derive a Spectrum from an RGB triplet.* IEEE Computer Graphics and Applications, July, 95–9, 1989.

GREENWALD, M.J., GREENWALD, S.L. and BLAKE, R., *Long lasting visual aftereffect from viewing a computer video display.* New England Journal of Medicine, 309, 315, 1983.

HUNT, R.W.G., *Measuring Colour,* Ellis Horwood, 1987.

ISHIHARA, S., *Ishihara's tests for Colour-Blindness,* 38 plate edition, Kanehara Ltd., Tokyo, 1983.

JAMESON, D. and HURVICH, L.M., *Theory of brightness and color contrast in human vision,* Vision Res. 4, pp.135–54, 1964.

KRAUSKOPF, J., ZAIDI, Q. and MANDLER, M.B., *Mechanisms of simultaneous color induction.* J. Opt. Soc. Am., 3(10), 1752–7, 1986.

LIVINGSTONE, M.S., *Art, Illusion and the visual system,* Scientific American, 68–75, 1982.

LONG, T., *Human factors principles for the design of computer colour graphics displays.* Br. Telecom Technol. J. 2(3), 5–14, 1984.

LONG, G.M. and GARVEY, P.M. *The effects of target wavelength on dynamic visual acuity under photopic and scoptopic viewing.* Human Factors, 30(1), 3–13, 1988.

MEYER, G.W., GREENBERG, D.P., *Colour defective vision and computer graphics displays.* IEEE Comp. Graphics and Appl., Sept, 28–39, 1988.

MICHAEL, C.R., *Columnar organisation of color cells in monkey's striate cortex.* J. of Neurophysiology, 46, 587–604, 1981.

MOLLON, J.D. and CAVONIUS, C.R., *Derivation of a uniform colour space from discriminative reaction times,* In Conference proceedings of IERE Colour in Information Technology and Visual Displays, University of Surrey, pp.27–31, March 1985.

MURCH, G.M., *The Effective Use of Color: Cognitive Principles,* TEKniques, 8(2), pp.25–31, 1984.

SAVOY, R.L., *Contingent aftereffects and isoluminance: psychophysical evidence for separation of color, orientation, and motion.* Computer Vision, Graphics and Image Processing, Vol 37, 3–19, 1987.

SCHWARTZ, M., COWAN, W. and BEATTY, J., *An experimental comparison of RGB, YIQ, LAB, HSV and opponent colour models.* ACM Transactions on Graphics, April, 1987.

SVAETICHIN, G. and MacNICHOL, E.F., *Retinal mechanisms for achromatic vision.* Annals of the New York Academy of Sciences, 74, 385–404, 1958.

TAJIMA, J. *Optical color display using uniform color scale.* NEC Res. and Dev., Vol 70, pp.58–63, 1983.

TYLER, C.W., *Analysis of visual sensitivity modulation.* J. Opt. Soc. Am. 'A', Vol 2, 1985.

TYLER, C.W., *Interlacing eliminates crt perceptible flicker.* Information Display, Vol 2, No 8, 1986.

UCHIKAWA, K. and BOYNTON, R.M., *Categorical colour perception of Japanese observers: comparison with that of Americans.* Color Vision, Vol 27, 1825–33.

VAN LAAR, D.L. and FLAVELL, R., *Towards a maximally contrasting set of colours.* In Jones, D. & Winder, R. Eds. 'Interacting with Computers IV'. pp.374–89, 1988.

VERRIEST, G., *Recent advances in the study of the acquired deficiencies of color vision.* Fondiazone 'George Ranchi', 24, 1–80.

VIVEASH, J.P. and LAYCOCK, J., *Computation of the resultant chromaticity coordinates and luminance of combined and filtered sources in display design.* Displays, Jan, 17–23, 1983.

WALRAVEN, J., *Perceptual artifacts that may interfere with colour coding on visual displays.* Proceedings of the workshop on colour coded vs monochrome electronic displays. Nato Report ds/a/dr84/431, 13.1–13.11, 1984.

WRIGHT, W.D., *The sensitivity of the eye to small colour differences.* Proceedings of the Physical Society (London), 53, 93–112, 1941.

WYSZECKI, G. and STILES, W.S., *Colour Science,* 2nd ed., Wiley, NY, 1982.

ZOLLINGER, H., *Categorical colour perception: influence of cultural factors on the differentiation of primary and basic colour terms and colour naming by Japanese children.* Vision Research, vol 28, no.12, 1379–82, 1988.

Appendices

Appendix A: Glossary

Brightness	The psychological or subjective impression of light intensity. (Adjectives: bright and dim.)
Cd/m^2	Candela per square metre, an objective measure of physical luminance.
Colour	The subjective sensation associated with light.
Hue	The attribute of a visual sensation, approximately synonymous with the common term colour, and described by terms such as red, green, yellow, blue and proportions of two of these.
Saturation	The purity, richness or vividness of a colour relative to its hue. A pink is an unsaturated red, Post-office red a highly saturated red. (Similar to colourfulness.)
Lightness	The brightness of an area judged relative to the brightness of a similarly illuminated area. (Adjectives: light and dark.)
Chromatic	Referring to the hue and saturation of a stimulus as opposed to the black-white or achromatic dimension.
Achromatic	Literally, 'without colour', but meaning without the sensations of hue or saturation, i.e. from black through greys of varying luminance to white.
Luminance	The physical intensity of light, correlated with but different from brightness (a psychological, subjective measure), measured in candelas per square metre cd/m^2.
Illumination	The amount of light falling onto a surface, measured in lux.
JND	Just noticeable difference – the difference between two stimuli that, under experimental conditions, is 'just noticeable', statistically speaking – discriminated as often as it is not discriminated.
Gamut	The space of possible colours which lie between, usually three phosphors on a colour monitor.
Fovea	The point on the back of the eye at which incoming light is focused.

Appendix B

CIE transformations

CIE 1931

$x = X/(X+Y+Z)$
$y = Y/(X+Y+Z)$
$z = Z/(X+Y+Z) = 1-(x+y)$

Y = luminance

CIE 1976 u', v'

$u' = 4x/(-2x+12y+3)$
$v' = 9y/(-2x+12y+3)$

CIE 1976 u, v hue angle and saturation

$h_{uv} = \arctan[(v'-v'_n)/(u'-u'_n)]$

where the actual value of h_{uv} depends on whether $v'-v'_n$ and $u'-u'_n$ are positive or negative, and u'_n and v'_n are values of u' and v' of the reference white.

$s_{uv} = 13[(u'-u'_n)^2+(v'-v'_n)^2]^{1M2}$

1976 Uniform colour space – L^*, u^*, v^*

CIE 1976 lightness

$L^* = 116(Y/Y^n)^{1/3} - 16$ when $Y/Y^n > 0.008856$
$L^* = 903.3(Y/Y^n)$ when $Y/Y^n \leqslant 0.008856$

The second formula is employed when the colour to be converted is quite dark.
$u^* = 13L^*(u' - u'_n)$
$v^* = 13L^*(v' - v'_n)$

note that

$h_{uv} = \arctan(v^*/u^*)$

CIE 1976 Colour difference formula

$\Delta E^*_{uv} = [\Delta L^*)^2 + (\Delta u^*)^2 + (\Delta v^*)^2]^{1/2}$

Where ΔL^*, Δu^* and Δv^* are the differences in L^*, u^* and v^* between the two colours being considered.

For further details see Hunt (1987) or Wyszecki and Stiles (1982).

Notes on the authors

C.W. Blatchford

Clive Blatchford is Manager, Secure Systems in the ICL Technical Directorate. In this capacity he has staff responsibility for definition, development and marketing of security in all ICL products and services.

He plays an active role in both National and International standards organisations and has lectured extensively in Europe, North America and Australia; he is currently Vice-Chairman of the USA Department of Commerce/National Bureau of Standards Special Interest Group on Security. More recently he has been involved in studies of security programmes for the European Commission. Prior to joining ICL he had senior management experience in dataprocessing and/or security with Xerox Corporation and with Chase Manhattan Bank.

John Cheesman

John Cheesman studied mathematics/computer science at the University of York, graduating in 1983, and joined ITT. He worked initially in software engineering on the System 12 tool chain, a major digital telecommunications switch, at a variety of sites throughout Europe. He then joined the ITT research centre at Harlow, working on the development and application of an in-house expert system. In 1988 he joined STC Technology, working in the Knowledge-Based Systems group, particularly on the requirements and design for the Strategic Marketing Information System (SMIS) discussed in the paper and employing the KADS technology.

A.A. Clarke

Anthony Clarke is a lecturer in the Department of computer studies at Loughborough University of Technology, and a Co-Director of the LUT-CHI research centre. Prior to joining the academic world as part of the Government's IT'83 programme he was group leader, Man-Systems Interface with Plessey Electronic Systems Ltd. Before 1978 he was Head of the Ergonomics Laboratory, EMI Ltd., and before 1973 he was Human Factors Section Leader with EASAMS Ltd., responsible for Tornado cockpit design aspects. He graduated with a BSc(Tech). Hons. in Occupational Psychology in 1970, following 15 years in the RAF and the aerospace industry, gaining aircraft and electrical engineering qualifications. He is a Fellow of the Ergonomics Society, an Associate Fellow of the British Psychological Society, and a Chartered Psychologist.

T.M. Cole

Terry Cole joined ICL in 1968 after graduating from Oxford University with a degree in Engineering Science. He has worked on George 3, VME/B and VME/K superstructure products, and then on the System Ten and its successor System 25. He has been involved with UNIX since the beginning of 1986 when this was ported to the System 25.

Christopher Dobbyn

Chris Dobbyn graduated from Keele University with honours in English and Philosophy in 1973 and received a post-graduate Certificate in Education from Birmingham in 1975. After working for ten years as a lecturer in English Literature in further education he took an MSc in computation at UMIST, specialising in Artificial Intelligence. For the past four years he has worked in the computing industry as a knowledge engineer and technical consultant, specialising in knowledge-based systems. He joined ICL Retail Systems in 1989.

Professor Ernest Edmonds

Professor Edmonds holds a PhD in mathematical logic from Nottingham University. He is Head of the Department of Computer Studies and Director of LUTCHI Research Centre at Loughborough University. His research has included work on graphics, speech input, computer-aided learning, computer communication and interactive dialogues. User interface software systems have been of central concern since the early 1970s.

He is the author of numerous papers in research topics, ranging from "An appraisal of some problems of achieving fluid Man/Machine interaction" to "Determining requirements and prototyping the User Interface Module". He is General Editor of the journal "Knowledge Based Systems" for the publishers Butterworth Scientific.

Dr Richard Flavell

Richard Flavell is a Reader at The Management School, Imperial College. He holds a BSc in Chemical Engineering and a PhD in Management Science, both from the University of London. He has taught at Imperial for the past 15 years, specialising in finance and project appraisal. He has published on these topics in a variety of journals. He has also worked in a bank dealing room and is a recognised financial trader.

Much of his work involves the production of computer-based financial models. Several years ago, this stimulated the question of how to produce effective displays, and especially using colour, for these models. Casting around, very little useful guidance was found and hence an interesting sideline was born. This work has been funded solely by industry over the past seven years and is becoming widely recognised as a leading source of advice.

Fiona Flower

Graduated from University of Aberdeen in 1978 with MA Honours in Modern Languages (French and Spanish).

Joined ICL in September 1978 as a graduate recruit, and completed the graduate induction programme.

First assignment with ICL was with Local Government and Public Corporations (LGPC) in Bristol, as an account support executive. After a short secondment to Dataskil in Reading, she was tasked with providing all pre- and post-sales support for 7700 for LGPC customers in Wales and the West Country. In 1981 she transferred to major Accounts, and provided accounts support in office systems for Major customers such as House of Fraser and Dowty.

When Customer Services was formed, she joined the Small Systems support team in Bristol, providing support in DRS systems and applications to customers in the Wales and West Country area, and remained with Customer Services until May 1985, when she took 6 months' maternity leave.

Following maternity leave, she joined CPS as a technical author, working part time from home. In the 4 years since then, she has completed a wide variety of projects, producing documentation for a number of clients, both internal and external. Internal clients included Defence Region, Office Systems, CRS&S, Networks Region, Northern Telecom and CIS. Her current project is to produce two of three booklets for Andrew Hutt, to complement his Marketing to Design lectures:

"How to Identify a High Valued Solution" (describing a process for deriving product attributes from user needs).

How to Deliver a Business Solution" (describing how to identify the manuals, man-machine interface and services to be delivered with a product).

Dr Andrew T.F. Hutt

Dr Hutt holds a BSc Mathematics from Durham University and PhD Computer Science from Southampton University and has worked for ICL since 1966.

At ICL between 1966 and 1973, he worked on the design of the Edinburgh Multi-Access system and of VME Operating System. In 1973, he was seconded to Southampton University where he developed an early Relational Database Management system. On his return to ICL he led a 25 man team which produced QueryMaster, Personal Data System and CAFS Relational Interface which are based on his research work. These products are used widely throughout the ICL customer base.

In 1984, Dr Hutt became ICL's HCI Strategy Manager responsible for coordinating all the HCI work in ICL. In this capacity he had led 2 Alvey projects: User Skills and Task Match and the Alvey Lab. Project: both of which have produced valuable reseach results. He has also contributed to other collaborations namely, the Alvey Early Evaluation Project, and the Esprit HUFIT and Eurohelp projects. Dr Hutt has been responsible for the industrialisation of the results of this research work and has developed the "Marketing to Design" human factors based methodology which is routinely used throughout ICL.

In the period 1985 to 1989, Dr Hutt has also worked on the MAP MultiStar Project on distributed relational database systems which has produced the only usable European system of this type.

H. Lesan

Hamid Lesan received his Ph.D. in Mathematical Logic from the Department of Mathematics of the University of Manchester in 1978. After a brief period of teaching, he joined STL in 1980, where his work has been mainly concerned with Formal Methods and Natural Language Processing. He is currently working on tools for supporting the formal specification and development of software in the context of the ESPRIT project RAISE.

T.A. Parker

Tom Parker graduated from Downing College, Cambridge, with a degree in mathematics in 1963. He first worked for Ferranti as a technician on a large in-flight simulator for the Bloodhound MK II missile before joining Fisons in 1966 as Systems Analyst and then Chief Programmer. He joined ICL in 1971 and worked on early design of major 2900 Series operating systems before transferring to Bureau West, where he was responsible for operating system support and for design and development of new security features. He is now a Security Consultant responsible for consultancy on all aspects of technical security within ICL.

He was responsible for the inception and for much of the design of the High Security Option of ICL's VME operating system, and is chairman of ICL's corporate Technical Security Strategy Subcommittee. He is actively involved in the design of high security network architecture within both ICL and standards organisations in Europe. He has written a number of technical papers on security and has lectured both in Europe and in America.

G. Poskitt

Geoff Poskitt joined ICL in 1978 to do the system design for System 25. In 1981 he was seconded to the Three Rivers Computer Corporation to work on graphics workstations. He returned to the U.K. in 1983 and worked on ICL's personal computer products, including the OPD and the ICL PC. For the past two years he has has been the systems architect and design authority for UNICORN.

Roger Pullen

Roger Pullen is Products Integration Manager within Office Systems Servers Product Group. He joined ICL in 1977 as a design/product engineer with the newly formed Product Development Group at Bracknell. This group was responsible for the electro/mechanical development on both System 10 and System 25 minicomputers. Over the years he has also gained considerable experience in working with the overseas agency approval test houses, such as Underwriters Laborities (UL), Canadian Standards Association (CSA) and Federal Communications Commission (FCC). Currently he manages the Products Integration Group responsible for UNICORN systems.

N.R. Seel

After working for several software houses where he specialised in systems programming and formal software specification, Nigel Seel joined STL in 1982. Having initially worked on formal specification support tools and Artificial Intelligence systems, in 1985 he became manager of the Intelligent Systems laboratory, developing "computer-based assistants". Two years later he was appointed technical manager of a major Alvey project, Adaptive Intelligent Dialogues, seeing the project through to its conclusion. He currently occupies a senior technical position in STL's Professional Community Support programme, specialising in Computer Supported Co-operative Working (CSCW). He is a graduate of the Open University, and has recently received his Ph.D. in formal cognitive science from Surrey University.

M. Smyth

Michael Smyth is a research assistant at the Loughborough University of Technology Computer Human Interface (LUTCHI) Research Centre. After graduation from The Queen's University of Belfast in 1983 with a B.A. (Hons) in Psychology, he completed a postgraduate course in Computer Science. In 1985 he joined Autographics Software (NI) as a programmer where he was part of a small team which specialised in the design and development of CAD products for education. Since joining LUCHI in 1987 he has been a member of the Alvey Project (MMI/062) Human Computer Cooperation. He is a member of the European Association for Cognitive Ergonomics (EACE).

Darren van Laar

Darren van Laar is a research assistant in Imperial College, London where he also lectures in Human-Computer Interaction (HCI). He is currently completing his PhD thesis on the use of colour with computers.

His first degree was in Psychology from Manchester Polytechnic in 1985 after which he studied computing and Human-computer interaction for his MSc. at York University. Since that time he has been working at Imperial College on various projects concerned with HCI, colour, statistics, computing and psychology.

ICL TECHNICAL JOURNAL

Guidance for Authors

1. CONTENT

The *ICL Technical Journal* has a large international circulation. It publishes papers of high standard having some relevance to ICL's business, aimed at the general technical community and in particular at ICL's users and customers. It is intended for readers who have an interest in the information technology field in general but who may not be informed on the aspect covered by a particular paper. To be acceptable, papers on more specialised aspects of design or applications must include some suitable introductory material or reference.

The Journal will usually not reprint papers already published, but this does not necessarily exclude papers presented at conferences. It is not necessary for the material to be entirely new or original. Papers will not reveal matter relating to unannounced products of any of the STC Group companies.

Letters to the Editor and reviews may also be published.

2. AUTHORS

Within the framework defined by §1 the Editor will be happy to consider a paper by any author or group of authors, whether or not an employee of a company in the STC Group. All papers are judged on their merit, irrespective of origin.

3. LENGTH

There is no fixed upper or lower limit, but a useful working range is 4000–6000 words; it may be difficult to accommodate a long paper in a particular issue. Authors should always keep brevity in mind but should not sacrifice necessary fullness of explanation to this.

4. ABSTRACTS

All papers should have an Abstract of not more than 200 words, suitable for the various abstracting journals to use without alteration. The Editor will arrange for each Abstract to be translated into French and German, for publication together with the English original.

5. PRESENTATION

5.1 Printed (typed) copy

Two copies of the manuscript, typed 1½ spaced on one side only of A4 paper, with right and left margins of at least 2.5 cms, and the pages numbered in sequence, should be sent to the Editor. Particular care should be taken to ensure that mathematical symbols and expressions, and any special characters such as Greek letters, are clear. Any detailed mathematical treatment should be put in an Appendix so that only essential results need be referred to in the text.

5.2 Diagrams

Line diagrams will if necessary be redrawn and professionally lettered for publication, so it is essential that they are clear. Axes of graphs should be labelled with the relevant variables and, where this is desirable, marked off with their values. All diagrams should have a caption and be numbered for reference in the text, and the text marked to show where each should be placed – e.g. "Figure 5 here". Authors should check that all diagrams are actually referred to in the text and that all diagrams referred to are supplied. Since diagrams are always separated from their text in the production process these should be presented each on a separate sheet and, most important, each sheet must carry the author's name and the title of the paper. The diagram captions and numbers should be listed on a separate sheet which also should give the author's name and the title of the paper.

5.3 Tables

As with diagrams, these should all be given captions and reference numbers; adequate row and column headings should be given, also the relevant units for all the quantities tabulated. Short tables can be given in the text but long tables are better submitted on separate sheets and these, as for diagrams, must carry the author's name and the title of the paper.

5.4 Photographs

Black-and-white photographs can be reproduced provided they are of good enough quality; they should be included only very sparingly. Colour reproduction involves an extra and expensive process and will be agreed to only exceptionally.

5.5 References
Authors are asked to use the Author/Date system, in which the author(s) and the date of the publication are given in the text, and all the references are listed in alphabetical order of author at the end.

e.g. in the text: "... further details are given in [Henderson 1986]"

with the corresponding entry in the reference list:

HENDERSON, P. Functional Programming, Formal Specification and Rapid Prototyping. IEEE Trans. on Software Engineering 1986, SE-12, 2, 241–250.

Where there are more than two authors it is usual to give the text reference as "[X et al ...]".

Authors should check that all text references are listed, and only text references; references to works not quoted in the text should be listed under a heading such as "Bibliography" or "Further reading".

5.6 Style
A note is available from the Editor summarising the main points of style – punctuation, spelling, use of initials and acronyms etc. – preferred for Journal papers.

6. REFEREES
The Editor may refer papers to independent referees for comment. If the referee recommends revisions to the draft the author will be asked to make those revisions. Referees are anonymous. Minor editorial corrections, as for example to conform to the Journal's general style for spelling or notation, will be made by the Editor.

7. PROOFS, OFFPRINTS
Printed proofs are sent to authors for correction before publication. Authors receive 25 offprints of their papers, free of charge, and further copies can be purchased; an order form for copies is sent with the proofs.

8. COPYRIGHT
Copyright in papers published in the *ICL Technical Journal* rests with ICL unless specifically agreed otherwise before publication. Publications may be reproduced with the Editor's permission, which will normally be granted, and with due acknowledgement.

Translations of Abstracts

G. Poskitt
Structure d'UNICORN

UNICORN est un système UNIX multi-utilisateur et multi-processeur mettant en oeuvre la technologie des ordinateurs à jeu d'instructions réduit (RISC). Des antémémoires permettent de tirer pleinement parti de l'extrême rapidité des processeurs. Elles facilitent le partage d'un sous-système de mémoire commun aux processeurs et la mise en mémoire tampon du trafic entrée/sortie.

Cet article décrit la structure de base du système, et en particulier les mécanismes de mise en mémoire et les moyens d'assurer la cohérence des différentes antémémoires entre elles et avec la mémoire centrale.

T. M. Cole
L'aspect logiciel d'UNICORN

Cet article décrit brièvement la version d'UNIX portée sur UNICORN. Il examine les ajouts qui ont été faits au système et en particulier les "chaînes étendues".

L'environnement offert par cette fonction peut être facilement parté sur des cartes de contrôle d'entrée/sortie qui supportent des modules de chaînes UNIX. Le système acquiert ainsi une plus grande flexibilité. En effet, certaines options comprises dans les modules de chaîne peuvent alors être renvoyées aux cartes de contrôle d'entrée/sortie à un moindre coût de mise en oeuvre.

R. Pullen
La conception électromécanique d'UNICORN

Cet article présente les principes de base de la conception électromécanique d'UNICORN. Il décrit brièvement le boîtier et les éléments qui le composent: armoire logique, unités de disques, systèmes de refroidissement, alimentation et interconnexions internes.

En fin d'article figure la liste des normes nationales et internationales auxquelles UNICORN répond.

A. T. F. Hutt and F. Flower
L'interface système-utilisateur: un défi pour les utilisateurs et pour les concepteurs d'applications

En l'espace de quelques années, l'interface système-utilisateur est devenu un élément essentiel de toute application. L'auteur décrit son évolution technologique, tant du point de vue de l'utilisateur que du concepteur d'applications. Il en conclut que si le premier peut espérer une plus grande facilité d'utilisation, le second doit en revanche s'attendre à de plus grandes difficultés de mise en oeuvre.

E. A. Edmonds
L'émergence de l'interface utilisateur séparable

L'auteur examine brièvement le sens du terme "interface utilisateur" qu'il conçoit comme un élément identifiable d'un système. Puis il dresse l'historique de l'interface utilisateur séparable et souligne l'intérêt croissant soulevé par les logiciels d'interface et les techniques employées pour leurs spécifications. Enfin, il envisage l'architecture des interfaces utilisateur séparables et les conséquences pratiques de leur utilisation.

C. Dobbyn *and* J. Cheesman
SMIS: un système expert destiné au traitement des données commerciales

Le projet ESPRIT P1098, dont l'objet est de définir une méthode de développement des systèmes experts, a débouché sur la création du système SMIS (Strategic Marketing Information System).

Le SMIS permet d'extraire des informations de différentes sources telles que des fichiers ou des bases de données, sans que la connaissance d'un langage d'interrogation ou de la structure interne des données consultées soit nécessaire. Les informations sont ensuite présentées de façon claire et intelligible. Outre ces qualités, il faut remarquer l'admirable conception du système.

L'auteur examine l'évolution de ce système et la référence que constitue la méthode KADS (Knowledge Acquisition Design, Structured). En conclusion, il en donne une appréciation globale et envisage l'avenir qui lui est réservé, comme à d'autres systèmes similaires.

H. Lesan *and* N. R. Seel
Une interface de dialogue pour programmation par contraintes

De nombreux actes de modélisation tels que l'élaboration de plannings de travail ou de feuilles de calculs s'apparentent à la création de phrases dans un langage formel soumise à certaines contraintes.
On est naturellement amené à hiérarchiser les opérations qui constituent cette création, par exemple la division d'un projet en sous-projets ou la consultation d'une base de données source. Comme l'a démontré Grosz, la structure dégagée permet d'organiser un dialogue en fonction des tâches effectuées.

L'auteur explique dans quelle mesure ce processus d'élaboration, perçu comme une recherche de solution coopérative, engendre un dialogue à initiative mixte. A chaque étape du dialogue, un arbre virtuel, ou organigramme, s'affiche sur un écran à haute résolution. Chaque arbre est constitué de phrases écrites en langage naturel qui permettent à l'utilisateur de formuler ses réponses, tout en fournissant ou demandant, le cas échéant, des informations complémentaires. Il est également possible de modifier l'objet du dialogue.

L'utilisateur se déplace dans l'organigramme au moyen d'une souris. Il peut en outre consulter à l'écran une transcription du dialogue et une représentation graphique du projet, même si ce dernier n'est pas définitif.

M. Coon
SODA: l'interface ICL destinée a l'accès aux documents ODA

Au cours de cet article, l'auteur se propose d'examiner la norme ODA (Office Document Architecture) et les recherches effectuées par ICL dans le cadre du projet PODA (Piloting Office Document Architecture). Ce projet, qui a joué un rôle déterminant dans le développement et l'application de la norme citée, consiste à choisir ce cadre normatif pour réaliser un échange expérimental de documents entre systèmes. Il justifie la mise au point d'un gestionnaire de mémoire dédié à la manipulation de documents ODA, accompagné d'une interface de programme d'application et d'utilitaires.

L'existence d'une interface publiée (qu'elle soit normalisée ou appartienne au domaine public) dédiée à la création et à la manipulation de documents ODA et dotée d'utilitaires de support implique que ces utilitaires et non l'interface elle-même imposent un format particulier pour les documents stockés.

M. Smyth *and* A. A. Clarke
Le processus de coopération au sein d'un groupe et la conception de systèmes coopératifs

Cet article compare la coopération observée au sein d'un groupe d'individus sur des projets abstraits, avec ou sans recours à l'informatique et celle qui résulte de la relation homme-machine.

L'auteur cite un certain nombre d'ouvrages où apparaît clairement le bénéfice de la coopération au sein d'un groupe. Il présente les mécanismes de coopération homme-machine qui ont fait l'objet d'une étude expérimentale à l'université de Loughborough (University of Technology), en collaboration avec ICL au titre du projet Alvey numéro MMI/062.

A cette occasion, des cas fictifs de coopération ont été imaginés. Le plus simple d'entre eux, que l'article examine, concerne la disposition de mobilier dans un bureau, sachant que certaines contraintes doivent être respectées.

En conclusion, l'auteur analyse les avantages et les limites des mécanismes étudiés dans la gestion de l'interaction homme-machine.

C. Blatchford
Exigences légales de la sécurité informatique — controle des acces

Les juristes restent divisés sur la définition de la violation d'un système informatique. Cette absence de consensus est d'autant plus préoccupante qu'en droit pénal anglais, le piratage, à savoir l'accès illicite à un système informatique, ne constitue pas une infraction.

Cependant, dès la prochaine session parlementaire, une proposition de loi devrait être à l'examen.

Quelle que soit la portée de cette loi qui vise à prévenir des actes de piratage toujours plus nombreux, des modifications techniques dans l'architecture et les modalités d'exploitation des systèmes d'information devront être effectuées.

Cet article examine les faiblesses, tant administratives que techniques, de l'architecture des systèmes qui, une fois surmontées permettront de renforcer l'efficacité de la législation en matière de protection informatique.

Cette analyse pourrait bien influencer fondamentalement la façon dont les concepteurs de systèmes évaluent la pertinence des solutions informatiques.

T. A. Parker
Normes garantissant la sécurité des interfaces destinées aux applications réparties

L'accès aux grands systèmes répartis soulève le problème de la sécurité de l'information. A la multitude des postes de travail s'ajoute celle des applications, et de façon plus générale, des constructeurs. Face à cet éclatement, l'identité et les droits d'accès des utilisateurs doivent être définis, puis communiqués et gérés afin d'assurer la sécurité des systèmes. Cet article présente les recherches effectuées dans ce domaine par les organismes de normalisation et par ICL dans le but d'élaborer des normes permettant de garantir la sécurité informatique.

R. Flavell *and* D. van Laar
L'utilisation de la couleur a l'écran (1) *Fondements physiologiques et physiques des couleurs*

L'utilisation de couleurs à l'écran permet certes d'améliorer l'interaction homme-machine et d'obtenir une image plus fidèle que celle produite par un simple procédé de codage monochrome.

Cependant, la connaissance du domaine plus général dans lequel l'utilisateur intervient constitue un atout important. C'est pourquoi deux articles seront consacrés à ce vaste sujet.

Le premier traite de l'utilisation des couleurs sur écran cathodique, de la production et de la mesure des couleurs, du fondement physiologique de la sensation de couleur et des effets visuels dus à l'emploi de couleurs à l'écran.

Quant au second article, il démontre qu'il est possible d'améliorer l'affichage en utilisant la couleur pour relier les informations affichées aux tâches de l'utilisateur. Il décrit par ailleurs différents procédés d'utilisation des couleurs à l'écran, ainsi que leurs avantages et leurs inconvénients. En conclusion, les facteurs physiques et cognitifs examinés au cours des deux articles sont rappelés et l'auteur ajoute quelques conseils adressés à ceux qui souhaiteraient tirer parti du contenu de ces articles.