ICL Technical Journal

Volume 2 Issue 4 November 1981

ICL Technical Journal

Contents

Volume 2 Issue 4

ICL Technical Journal

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

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

Annual subscription rate: £10 (cheques should be made out to 'Peter Peregrinus Ltd.', and sent to Peter Peregrinus Ltd., Station House, Nightingale Road, Hitchin, Herts, SG5 1RJ, England. Telephone: Hitchin 53331 (s.t.d. 0462 53331).

Publisher
Peter Peregrinus Limited
PO Box 8, Southgate House, Stevenage, Herts SG1 1HQ, England

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

Architecture of the ICL System 25

Alan Walton
ICL Distributed Systems Division, Bracknell, Berks., UK

Abstract

The organisation and operation of the hardware of the ICL System 25 small-business machine is described. The relation to the existing System Ten, with which it is compatible, is indicated.

1 Introduction

System 25 is a physically small, powerful and versatile computer aimed primarily at the needs of commercial data-processing. It is compatible with the ICL System Ten which has proved a very popular and successful small-business machine and of which approaching 10,000 are in use world-wide. System Ten was designed in 1968, since which time there have been considerable advances in both physical technology and concepts of computer architecture. The design of System 25 exploits these advances to give a machine which is in every sense more powerful and versatile than System Ten while retaining all the good features of the earlier machine, especially its ease of use. In essence, the aim of the new design is to provide in a single small system, and simultaneously if required, all the various services which are being requested in business operations, such as batch, transaction and distributed processing, word processing, and control of such devices as Point-of-Sale and Factory Data Collection terminals.

The machine is shown in Plate 1.

Particular features of System 25 architecture are as follows:

(i) Features derived from the Primitive Level Interface, which is the definition of the prime interface between the software and the hardware and includes the instruction set, store map, data and arithmetic standards etc. It is based on that for System Ten with extensions for handling 8-bit data and new high-capacity discs, maintaining exact compatibility wherever possible so as to allow direct transfer of programs. The main features here are:

   (*a*) Partitioned store, to allow multi-programming with guaranteed independence and protection for the programs
   (*b*) Decimal arithmetic and decimal addressing; this allows computation on input data without the need for time-consuming decimal-to-binary conversion, and also makes the use of the machine easier and more natural.
   (*c*) Variable-length store-to-store operations, to allow efficient use of main-store space.

(ii) Support for a number of communication protocols, allowing System 25 to be used as a secondary station to ICL and IBM mainframes or as a primary station driving remote peripherals.

(iii) Means for supporting a wide range of peripherals via either System 25 standard interfaces (carried over from System Ten) or various Industry Standard interfaces. The main slow peripheral interface, for example, will allow the connection of up to 10 peripherals at distances of up to 1500m.

(iv) The System 25 Bus, which is the key feature of the hardware organisation and on which the above features depend. It is a set of physically separate bussed highways, used to interconnect the modules which make up the system. Each highway is designed for its particular purpose, resulting in cost-effective system modules and efficient transfer of data within the system; and the architecture provides ample scope for the incremental introduction of future enhancements.

The present paper describes in some detail the organisation and operation of the hardware and the means by which these and other features are realised. It concludes with a short note on the main software packages which are provided for support of the hardware; these will be dealt with in more detail in a second paper, to be published in the next issue of this Journal.

2 Hardware Organisation

2.1 Summary

The key to the System 25 hardware organisation is a set of bussed highways collectively known as the System 25 Bus, which is used to interconnect a number of modules to create a system as shown in Fig. 1.

A system contains at least one of the following system modules, each of which consists of one or more boards interconnected via the backplane containing the System 25 Bus:

- a Control Processor which is responsible for the overall supervision of the system and for data transfers from slow peripherals
- an Instruction Processor which executes the System 25 Instruction Set
- a Store module
- a Disc Controller plus its associated Disc Adaptor which is responsible for fast I/O transfers
- a Slow Peripheral Coupler to drive, an external slow peripheral interface for the connection of peripherals such as VDUs, printers.

There are two external slow peripheral interfaces supported by System 25, which have been carried over from System Ten. The main slow peripheral interface is the MTIOC interface (Multi Terminal Input Output Channel), which allows connection of up to ten slow peripherals onto a single twisted pair. The MDIOC interface

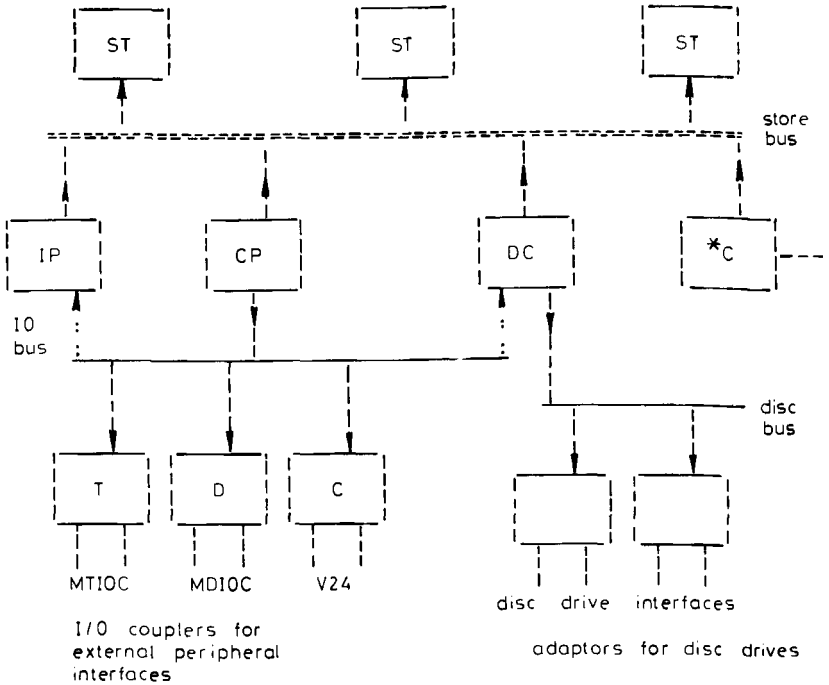


Fig. 1 Basic structure of System 25

Notes
(i) Arrows indicate master/slave relationship
(ii) Dotted lines indicate control information only
(iii) MTIOC etc. explained in text

Key
ST = Store Modules
IP = Instruction Processor
CP = Control Processor
DC = Disc Controller
*C = Other Controllers
T = T Coupler - MTIOC Interface
D = D Coupler = MDIOC Interface
C = C Coupler - Communication Lines

(Multi Device Input Output Channel) allows connection of Point-of-Sale and Factory Data Collection terminals carried over from System Ten.

System 25 also supports a number of local disc drive interfaces which are specific to the particular disc drive; and a variety of communications lines and their associated protocols.

2.2 System 25 Bus

The System 25 Bus consists of two main bussed highways – the Store Bus and the I/O Bus – and a subsidiary Disc Bus. There are two lines common to all modules; a

System Reset line and a System Fail line.

The System 25 Bus is tracked in the back plane. The common lines, Store Bus and I/O Bus are tracked to all board locations. The Disc Bus is only tracked to those locations which house a Disc Controller and its associated Disc Adaptors.

On any of the highways, transfers across the highway can only be initiated by a master module. The other module involved in the transfer is a slave module.

2.2.1 Store Bus: The Store Bus is used to transfer either one or five bytes of data between one of the master modules and one of the store modules (a slave module) and consists of

- 23 Address lines (7 bit binary Block Address and a 4 digit decimal Byte Address within a block)
- 40 Bidirectional Data lines (5 bytes)
- 2 Store Function lines
- 1 Store Mode line
- 3 Interface Control lines
- 1 Master Clock line (4 MHz, equivalent to a clock period of 250nsec)

Transfers on the Store Bus occur in synchronism with the Master Clock.

A 'daisy-chain' Bus Request line is used to resolve contention for use of the Store Bus. When a master module obtains access to the Store Bus, it will set Bus Reserved to inhibit the contention logic for one slot time. At the same time the master module enables the address and data onto the bus lines. The store module will respond with data in time to be latched into the master module at the end of the second slot. Thus a single transfer takes two clock periods.

During the second slot contention can take place for the next store cycle.

The store is defined to have two modes of operation. In Word mode, 5 bytes of data are transfered in parallel across the bus. In Byte mode, a multiplexor in the store module is used to transfer a single byte of data on the lowest numbered byte data lines.

The store cycle is 500ns (two clock periods) giving bandwidth of 10Mbyte/s in Word mode and 2Mbytes/s in Byte mode.

2.2.2 I/O Bus: The I/O Bus is driven by the Control Processor (the master module) and is used as the transfer path for control information to other modules and as a byte multiplexing interface for the transfer of data to or from I/O Couplers. It consists of

- 8 Bidirectional data lines
- 2 Function lines
- 2 Function Qualifier lines
- 5 Interface Control lines
- 1 Strobe Line

The transfer of a single byte of data takes a minimum of 8 interactions on the I/O Bus.

The I/O Bus provides for up to 64 Input/Output Channels (IOCs) which are independent logical transfer paths, a single coupler may have more than one IOC.

Each IOC is defined to have a set of 4 output registers which are written to by the Control Processor and a set of 4 input registers which are read by the Control Processor. Access to these registers is controlled by the Function and Function Qualifier lines.

The Control Processor can select a particular IOC by writing the IOC number to one of the output registers (Come On Line) and the selected IOC will then remain 'On Line' until 'Offline' is set or another IOC is selected.

The Control Processor will initiate the transfer of a block of data by writing to the Control Word register of the selected IOC. Thereafter the transfers of the command, data and status bytes for the block of data are initiated by an interrupt from the IOC.

There is a single bussed interrupt line which is set by an IOC requesting service. A 'daisy chained' Interrupt Acknowledge line will cause the highest priority interrupting IOC to place its IOC number on the data lines.

The Control Processor will bring the interrupting IOC Online and read from the input registers the interrupt type, associated data and the assigned partition number. The Control Processor will then write any output data to the output registers. Finally, the Control Processor will clear the interrupt by writing to the Clear Interrupt register and set the IOC Offline.

2.2.3 Disc Bus: The Disc Bus is driven by the Disc Controller and is used as the transfer path for control information to Disc Adaptors, and as a block multiplexing interface for the transfer of data to or from Disc Adaptors. It consists of

- 8 Bussed Bidirectional Data lines
- 1 Bussed Control line
- 1 Select line to each Adaptor
- 1 Service Request line from each Disc Adaptor
- 1 Strobe line.

The Disc Bus consists of one or more physically separate segments, each driven by one Disc Controller.

Each Disc Adaptor contains a buffer which is used to store the disc transfer parameters sent across the data lines by the Disc Controller.

When the Disc Adaptor is ready to transfer a block of data, it requests service from the Disc Controller. The Disc Controller will initiate a data transfer operation and the block of data is transferred across the disc bus in synchronism with the Strobe line at 3Mbytes/s.

2.3 *External Peripheral Interfaces*

2.3.1 MTIOC Interface: The MTIOC interface (Multi Terminal Input Output Channel) is the main interface for connecting slow peripherals onto System 25, and has been carried over from System Ten. Up to 10 terminals may be multi-dropped onto a single twisted pair cable which can be up to 1500m long.

Data is transmitted bit-serially as 7 data bits and two check bits at a bit rate of 28kHz or 56kHz, using frequency modulated encoding. The equivalent byte rate is nominally 1500 or 2700 characters transferred per second. Successful transmission of a byte of data in one direction is followed by an Acknowledge transmission in the other direction. If no acknowledgment is detected, the data byte is automatically retransmitted.

The protocol of the MTIOC interface allows for four commands:- Read, Write, Read Control and Write Control. Interpretation of the commands depends on the particular peripheral. In between transfers, the terminals are polled in sequence. The selected terminal will acknowledge the poll if it requires service.

Electrical isolation of the interface is achieved by transformer coupling at the terminal.

2.3.2 MDIOC Interface: The MDIOC interface (Multi Device Input Output Channel) is used for connecting Point-of-Sale or Factory Data Collection terminals into System 25 via the D coupler (see Fig. 1). It allows the connection of a single terminal up to 12000m from System 25 via a single twisted pair. The data transfer rate depends on the direction of the transfer and length of line and can be set to the following data rates (in characters per second).

Output to Terminal	Input from Terminal	Maximum Line Length
120	1200	12000m
1200	3600	6000m
2400	4800	1000m

Data is transmitted as 7 data bits and one parity bit using an asynchronous transmission protocol. Optical couplers in the D Coupler give electrical isolation. The terminal provides the line current which is modulated by the terminal for input data, and the impedance at the D Coupler is modulated for output data.

The coupler contains a multiplexor for the connection of up to 16 lines into a single IOC, and two IOCs with a total connection capability of 32 lines.

2.3.3 Communication Interfaces: CCITT V24 interfaces are provided for the connection of modems and associated communications lines. System 25 will support a variety of protocols, including:

ICL CO1	
ICL CO3	binary synchronous protocols
IBM 3270	
IBM SNA	SDLC
X25	HDLC

There is a special binary synchronous protocol for the connection of remote System 25 or System Ten video terminals.

2.3.4 Other External Interfaces: Other external interfaces can be supported by the use of special-purpose couplers, for example:

RS 232 interface for special purpose printers.
Cartridge Magnetic Tape Drive interface.

2.4 System Modules

2.4.1 Instruction Processor: The Instruction Processor is a master module on the Store Bus and a slave module on the I/O Bus. It is responsible for execution of the System 25 Order Code.

Instruction execution is initiated by commands on the I/O bus from the Control Processor and continues until an I/O instruction is encountered or it is commanded to switch to another partition by the Control Processor. For I/O instructions, the relevant real addresses are computed and stored in the control store and the Control Processor is informed by an interrupt on the I/O Bus.

When commanded to switch partition it will continue processing until the next successful branch instruction before suspending operations and informing the Control Processor.

2.4.2 Control Processor: The Control Processor is *a* master module on the Store Bus and *the* master module of the I/O bus. It is responsible for the overall control of the system and for transferring data between Slow Peripherals (via the I/O Bus) and main store (via the Store Bus).

It initiates processing by the Instruction Processor and controls switching between partitions.

It initiates slow peripheral operations by translating the I/O instruction into commands which are then routed to the appropriate coupler. It maintains the store address and counts associated with the transfer, and operates as a byte multiplexor for the transfer of data between the I/O Bus and Store Bus. It initiates transfers to fast peripherals which then proceed autonomously under the control of a disc controller. At the end of the transfer the Control Processor will initiate a partition switch to the originating partition.

It uses a configuration table in the control store to translate logical device numbers into real device numbers.

2.4.3 Store Modules: A store module is a slave on the Store Bus and provides a five-byte-wide random access store. It will generate parity when writing to store, and check the stored parity when reading from store.

2.4.4 Disc Controllers: A disc controller is *a* master module on the Store Bus, is *a* slave module on the I/O Bus and is *the* master module of a Disc Bus.

It is responsible for scheduling disc operations and for transferring data between Disc Adaptors (via the Disc Bus) and Main Store (via the Store Bus). It initiates disc operations at the request of the Control Processor by translating commands contained within the Control Block associated with a Disc I/O instruction and routing the information to the appropriate Disc Adaptor.

It maintains the store address associated with the transfer, and operates as a block multiplexor for the transfer of data between the Disc Bus and Store Bus.

A Disc Adaptor is responsible for interfacing the disc drives into System 25 and for transforming the protocols and data formats used within System 25 into commands and data across the disc drive interfaces. There is a different Disc Adaptor for each type of disc drive.

2.4.5 Slow Peripheral Couplers: A Slow Peripheral Coupler is a slave on the I/O Bus and controls one or more external peripheral interfaces of communications lines. It is responsible for driving the external interface and for transforming the protocols and data formats used by the peripherals or communications lines into those used within System 25.

Each Slow Peripheral Coupler consists of one or more logical Input Output Channels (IOC). There is a different Slow Peripheral Coupler for each type of external interface as follows:

T Coupler for MTIOC interfaces
D Coupler for MDIOC interfaces
C Coupler for communications lines
R Coupler for Cartridge Magnetic Tape
V Coupler for special interfaces

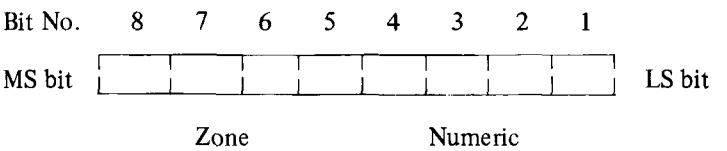
3 System 25 Primitive Level Interface

The Primitive Level Interface of a system is the main interface between the hardware of the system and the software running in the system. It consists of a definition of the Instruction Set plus a definition of associated data standards, store map, arithmetic standards, etc.

The System 25 Primitive Level Interface is based on that of System Ten, with a limited number of extensions to support the new high capacity discs, to improve performance and to handle 8-bit data.

3.1 Data Standards

3.1.1 Data Codes: Data is stored within System 25 as 8-bit bytes as follows:



The bits are numbered from 1 to 8; bit 8 is the most significant bit and bit 1 the least. Bits 8 to 5 contain the Zone code of the byte and have a value in the range 0 to 15. Bits 4 to 1 contain the Numeric code of the byte and have a value in the range 0 to 15.

The characters assigned to the codes are given in Table 1 for Zone codes 0-7 and correspond to the ISO 7 bit set. Zone codes 8-15 do not have assigned characters.

The 6 bit subset which is compatible with System Ten consists of Zone codes 2-5 (i.e. bits 8 and 6 are ignored).

3.1.2 Numeric Data: Numeric data is stored as variable length Binary Coded Decimal (BCD) byte strings in sign and modulus form. The Numeric code of each byte has a value in the range 0-9. For positive numbers each byte has a Zone code of 3, so that numbers are stored as their character code.

For negative numbers each byte except the least significant byte has a Zone code of 3. The least significant byte has a Zone code of 5. Thus the character representation of the least significant byte for the numerals 0 to 9 is P to Y, as in the following table:

Numeral	0	1	2	3	4	5	6	7	8	9	Positive number
Character	P	Q	R	S	T	U	V	W	X	Y	Negative number

Example: -1234 is stored as 123T.

3.2 Store Map

The Main Store is divided into a number of areas as shown in Fig. 2.

(*a*) Control Store from 0 to 2999, which contains I/O control and configuration data for use by the Control Processor.

(*b*) Common Store from absolute address 3000 which contains information accessible to all partitions. Write access is prohibited to locations 0 to 299 of Common. This area is called Protected Common.

(*c*) Partition Store. An area of store for each partition which contains the partition program, data and Index Registers.

Table 1 **Character codes**

					Column 0	1	2	3	4	5	6	7	Bit	
					0	0	0	0	0	0	0	0	8	
					0	0	0	0	1	1	1	1	7	ZONE CODE
					0	0	1	1	0	0	1	1	6	
Bit	4	3	2	1	0	1	0	1	0	1	0	1	5	
	0	0	0	0	NUL	DLE	SP	0	@	P	`	p		
	0	0	0	1	SOH	DC1	!	1	A	Q	a	q		
	0	0	1	0	STX	DC2	"	2	B	R	b	r		
	0	0	1	1	ETX	DC3	£	3	C	S	c	s		
	0	1	0	0	EOT	DC4	$	4	D	T	d	t		
	0	1	0	1	ENQ	NAK	%	5	E	U	e	u		
	0	1	1	0	ACK	SYN	&	6	F	V	f	v		
	0	1	1	1	BEL	ETB	'	7	G	W	g	w		
	1	0	0	0	BS	CAN	(	8	H	X	h	x		
	1	0	0	1	HT	EM	)	9	I	Y	i	y		
	1	0	1	0	LF	SUB	*	:	J	Z	j	z		
	1	0	1	1	VT	ESC	+	;	K	[	k	{		
	1	1	0	0	FF	FS	,	<	L	\	l	|		
	1	1	0	1	CR	GS	-	=	M	]	m	}		
	1	1	1	0	SO	RS	.	>	N	^	n	~		
	1	1	1	1	SI	US	/	?	O	_	o	DEL		

3.2.1 Control Store: The lowest numbered area of main store is called the Control Store which has been added for System 25. This area of store is not accessible to the partitions, except from Partition 0 in special circumstances, and is used to hold control information for Input/Output operations and for general system operation.

A duplicate set of A and B registers hold the control words in absolute address form and other registers are used by the Control Processor to progress the state of I/O transfers.

An area of control store is used to hold a translation table for the mapping of the logical device numbers within an I/O Instruction into real device numbers and IOC numbers.

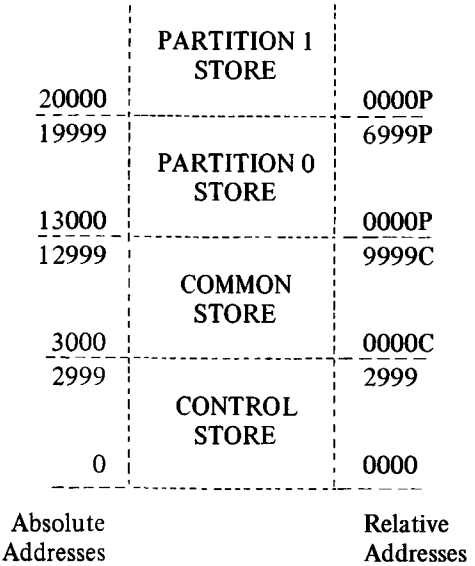


Fig. 2 Main store layout

3.2.2 Common Store: There is an area of Main Store accessible to all programs called Common. The size of Common is set during Initial Program Load to a value in the range 10000 to 80000 bytes, in units of 1000 bytes.

The first 300 bytes are called Protected Common and write access is inhibited. It is used to hold the Program Pointer (P register – 5 bytes) and Input/Output Control Words (A and B registers – 5 bytes each) for each partition.

The next 700 bytes are used for entry points to supervisory routines, tables controlling shared resources (such as disc), and a Mailbox for communication between partitions.

The remainder of Common is used to hold shared routines (e.g. housekeeping routines) and buffer areas for the bulk transfer of data between partitions.

3.2.3 Partition Store: Each program in System 25 operates within its own area of main store called a Partition and the System 25 architecture allows for up to 20 partitions. The size of each partition is set during Initial Program Load to a value in the range 0 to 80000 bytes, in units of 1000 bytes.

There is no defined structure within the partition store and its use is determined by the user program. However, bytes 11-14, 21-24, 31-34 are used as index registers by the instructions, and bytes 40-44 are used to hold the contents of the P Register when a Program Check occurs.

3.3 Arithmetic operations

Arithmetic operations use two operands called the A operand and the B operand,

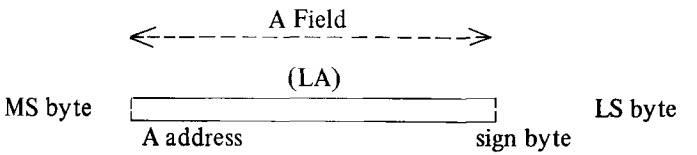
which are fetched from the A and B Fields. The result is stored in the B Field.

The basic arithmetic functions are Add, Subtract, Multiply and Divide, with these conventions:

Add – the A Operand is added to the B Operand.
Subtract – the A Operand is subtracted from the B Operand.
Multiply – the A Operand is multiplied by the B Operand.
Divide – the B Operand is divided by the A Operand.

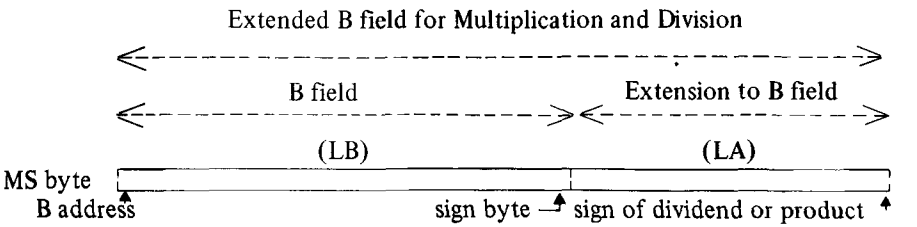
3.3.1 Numeric Fields: Numeric data is stored in variable length fields of up to 10 bytes.

The A Operand is contained within the A Field as shown below



The length of the A Field is defined by the contents of the LA field in the instruction.

The B Operand is contained in the B Field or the Extended B Field as shown below



The length of the B Field is defined by the contents of the LB Field in the instruction and the length of the extension to the B Field is defined by the contents of the LA Field.

3.3.2 Addition and subtraction: Addition and subtraction of numeric data is performed digit by digit starting with the least significant bytes of the two operands. The result of each byte is stored before the next more significant bytes are accessed.

The arithmetic operations performed on the two numeric parts of the bytes depend upon the Zone codes of the least significant bytes as defined in the table below. Zone codes 0-3 and 8-11 are treated as a positive, codes 4-7 and 12-15 are treated as negative. The zone code of the result is set to 3 or 5.

Function	B operand	A operand	Operation	Result	
				Sign	Zone Code
ADD	+	+	B + A	+	3
	+	-	B - A	+	3
	-	+	B - A	-	5
	-	-	B + A	-	5
SUBTRACT	+	+	B - A	+	3
	+	-	B + A	+	3
	-	+	B + A	-	5
	-	-	B - A	-	5

If the value of A is greater than the value of B, and the operation is B - A, then the sum is complemented and the sign of the result is inverted.

If the A operand is shorter than the B operand, it is effectively extended with zeros on the left during the operation.

If the A operand is longer than the B operand, it is truncated from the left and the truncated part is ignored.

3.3.3 Multiplication: Multiplication is performed by a sequence of additions of the A operand, extended with a zero digit, to the extended B field. The number of additions is determined by the value of the digits of the B operand.

The zone of the least significant byte of the extended B-field is set according to the signs of the A and B operands. Zone codes 0-3 and 8-11 are treated as positive signs.

Operand Signs		Result	
A	B	Sign	Zone Code
+	+	+	3
+	-	-	5
-	+	-	5
-	-	+	3

The result of using overlapped operands is undefined.

3.3.4 Division: Division is performed by a sequence of subtractions of the A operand (divisor), extended with a zero digit, from the extended B field which contains the B operand (dividend).

The result (quotient) is stored in the B field and the remainder is stored in the extension to the B field.

The Zone code of the least significant byte of the B field is set according to the signs of the A and B operands.

Operand Signs		Quotient	
A	B	Sign	Zone Code
+	+	+	3
+	-	-	5
-	+	-	5
-	-	+	3

The sign of the B operand is not changed and becomes the sign of the remainder.

The result of using overlapped operands is not defined.

3.4 Address format

The addresses on System 25 are represented as decimal numbers in the range 00000 to 79999, the first digit being a page number in the range 0 to 7. The whole address is stored in part of a 4-byte address word, together with a marker C which takes the value 0 if the address is relative to a partition base and 1 if it is relative to the Common base.

The format is as follows:

The decimal digits d3, d2, d1, d0 giving the address within the page with d3 the most significant (thousands) and d0 the least significant (units), are binary-coded in bits 4 to 1 of the four bytes. Bit 4 is the most significant, bit 1 the least.

The page number p is stored in inverse form in bit 5 of each of bytes 1, 2, 3 with the most significant bit in byte 3.

The marker C is stored in bit 7 of byte 4 and its inverse in bit 6.

The remaining bits – that is, bits 6 to 8 of bytes 1 to 3 and bit 5 of byte 4 – are not used by address computation.

The layout is shown in the diagram below.

Byte	0	1	2	3
Bit 8	*	*	*	0
7	*	*	*	C
6	*	*	*	$\overline{C}$
5	$\overline{P0}$	$\overline{P1}$	$\overline{P2}$	*
4				
3	D3	D3	D1	D0
2				
1				

*not used

The convention used here and in the following paragraphs is that a capital letter indicates a field and the corresponding lower-case letter the content of that field: thus *p* is the number of the page stored in the page-number field P, and the sub-field D3 of the address field contains the decimal digit d3.

3.5 Instruction format

3.5.1 General description: System 25 uses 2-address instructions of the form

F, A, B

where F is the function code and A,B are operand addresses. The interpretation of the contents of the address fields depends on the contents of other fields stored in the instruction word, as follows:

LA, LB	give the lengths of the operands
AC, BC	are address markers
IA, IB	concern indexing
EIX	concerns extended indexing
IDA, IDB	concern direct/indirect addressing

These are explained in the succeeding paragraphs.

An instruction occupies 10 bytes, which we number conventionally 0 to 9 from the left. The A,B addresses, in binary coded decimal, are held in bytes 1 to 4 and 6 to 9, respectively, with the format described in Section 3.4. The decimal digits are coded in bits 4 to 1 of the respective bytes, with bit 4 as the most significant bit. They must be in the range 0 to 9; a value in the range 10 to 15 will cause a Program Check.

The detailed layout of the instruction is shown in the diagram below, where the columns are the bytes and the rows the bits within the bytes. Bit 8 of each byte is always zero and bit 6, which normally contains the inverse of bit 7, is ignored.

Byte		0	1	2	3	4	5	6	7	8	9
Bit	8	0	0	0	0	0	0	0	0	0	0
	7	F3	F2	F1	F0	AC	IA1	IA0	IB1	IB0	BC
	6	*	*	*	*	*	*	*	*	*	*
	5	$\overline{\text{IDA}}$	$\overline{\text{PA0}}$	$\overline{\text{PA1}}$	$\overline{\text{PA2}}$	$\overline{\text{F4}}$	$\overline{\text{IDB}}$	$\overline{\text{PB0}}$	$\overline{\text{PB1}}$	$\overline{\text{PB2}}$	$\overline{\text{EIX}}$
	4										
	3	LA	A3	A2	A1	A0	LB	B3	B2	B1	B0
	2										
	1										

3.5.2 F, the Function: The function field F comprises 5 bits F0 to F4 located as follows:

F3 to F0 in bit 7 of each of bytes 0 to 3 respectively
F4, inverted, in bit 5 of byte 4

3.5.3 A and B, the Addresses (cf. Section 3.4): An address is specified by the combination of the P field (bits 5 of bytes 1-3 or 6-8) and four decimal digits (binary coded in bits 4-1 of bytes 1-4 or 6-9). The P field gives the page number in the range 0-7, stored in inverse form, and the decimal digits the adress of a byte within a page.

For the A address for example the P field is stored as follows:

bit 5 byte 1	$\overline{PA0}$	1	0	1	0	1	0	1	0
2	$\overline{PA1}$	1	1	0	0	1	1	0	0
3	$\overline{PA2}$	1	1	1	1	0	0	0	0
page number		0	1	2	3	4	5	6	7

and similarly for B.

In the decimal address, A3 is the most significant digit (i.e. the thousands digit) and A0 the least (i.e. the units digit).

3.5.4 LA and LB, the Operand Lengths: These fields are normally used to define the lengths of the operands in bytes; their contents can have values in the range 0 to 9, with the value 0 indicating a length of 10 bytes.

For some instructions the two fields are used together to give a single operand length in the range 1 to 100 bytes; in this case LA is the tens digit and LB the units, with 00 indicating a length of 100 bytes.

3.5.5 AC and BC, the Address Markers: These are used to indicate whether the address is relative to Common base or to a partition base. If AC is set to 1, the A address is in Common; if to 0, it is in the partition store and therefore relative to the current partition base.

3.5.6 IA, IB and $\overline{EIX}$: Indexing: The IA, IB fields are used to specify whether or not the relevant addresses are to be indexed and if so, which index registers are to be used. For the A address the interpretation is as follows:

IA1	IA0	Index Register
0	0	No indexing
0	1	11P to 14P
1	0	21P to 24P
1	1	31P to 34P

and similarly for B. The location of the index registers is given in Section 3.2.3.

The $\overline{EIX}$ field concerns *extended indexing*, explained in Section 3.6.1 below. This is a 1-bit field, stored in inverse form in bit 5 of byte 9. If this bit is set to 0, extended indexing is specified.

3.5.7 $\overline{IDA}$ and $\overline{IDB}$: Indirect Addressing: These fields are used to indicate whether the contents of an address field (possibly indexed) are to be used as the operand

address directly, or whether they point to another 4-digit field which contains the required address in relative address format but not aligned to a boundary.

The instruction contains the values of the field in inverse form; the interpretation is

$\overline{IDA}$ = 1 indicates direct addressing for A
$\overline{IDA}$ = 0 indicates indirect addressing for A

and similarly for B. The operation is explained further in Section 3.6.

3.6 Address Computation

3.6.1 Operand Relative Address: The relative address of the A Operand is evaluated in two stages as follows:

First, if indirection is specified ($\overline{IDA}$ = 0), the A field in the instruction (PA, A3, A2, A1, A0 and AC) is used to fetch the four byte field containing the indirect address, which is then used in place of the A field.

Second, if indexing is specified (IA ≠ 0), the contents of the specified index register are added.

If extended indexing is not specified ($\overline{EIX}$ = 1), the addition is performed modulo 10000.

If extended indexing is specified ($\overline{EIX}$ = 0), the addition is performed modulo 80000.

3.6.2 A Operand Address Marker, ac: The address marker of the A operand, ac, is evaluated in two stages in parallel with the evaluation of the operand relative address as follows:

First, if indirection is specified ($\overline{IDA}$ = 0), bit 7 of byte A+3 (of the indirect address field) is fetched and used in place of the AC field.

Second, if indexing is specified (IA ≠ 0) and extended indexing also is specified, bit 7 of the least significant byte of the specified index register is combined (logical OR) with the AC field to generate the address marker.

3.6.3 A Operand Absolute Address: If ac = 0, the A operand is in Partition and the A operand absolute address is formed by adding the A operand relative address to the partition base address

If ac = 1, the A operand is in Common and the A operand absolute address is formed by adding the A operand relative address to the Common base address (3000).

3.6.4 B Operand Address: The B operand address is formed in a similar manner using the PB, B3, B2, B1, B0 and BC fields in conjunction with IDB and IB.

3.7 Instruction Set

	Mnemonic		Instruction Name	Function Code	
Computational Instructions	A		Add		4
	S		Subtract		7
	M		Multiply		6
	D		Divide		5
	C		Compare		14
	IC	*	Indirect Length Compare		30
	–		Logical Instructions:	LA	20
	AND	*	– And	3	
	OR	*	– Or	1	
	NEQ	*	– Not Equivalent	2	
Address Arithmetic Instructions	MA		Move Address		3
	–		Modify Address Instructions:	LA	2
	AAI		– Add Address Immediate	0	
	AA		– Add Address	1	
	SAI	*	– Subtract Address Immediate	2	
	SA	*	– Subtract Address	3	
	CA	*	– Compare Address	4	
Data Transfer Instructions	MC		Move Character		8
	IM	*	Indirect Length Move Character		24
	MN		Move Numeric		9
	FN		Form Numeric		13
	X		Exchange		15
	PK	*	Pack		31
	UPK	*	Unpack		29
	E		Edit		12
Control Instructions	B		Branch		11
	SM		Set Mode		10
Input/Output Instructions	R		Read		0
	W		Write		1
	SR	*	Start Read		16
	SW	*	Start Write		17
	TS	*	Test IO Status		18

*New Instructions for System 25

4 Input/Output Operations

4.1 General:

System 25 Input/Output normally operates in a strictly synchronous manner in

that when a Read or Write Instruction is encountered, the partition is suspended until the I/O transfer is complete.

However, use of the Start Read and Start Write instructions will allow the partition to continue processing until the next I/O instruction is given. Use of the Test IO Status instruction allows the progress of the I/O transfer to be monitored.

In System 25, peripherals are divided into two types:-

slow peripherals which are assigned to a single partition and can only be driven by that partition.

fast peripherals which are shared between all partitions and can be driven by any partition.

The System 25 I/O instructions provide direct control of all the peripherals assigned to a partition, although disc transfers are normally controlled by invoking supervisory routines.

4.2 Slow Peripheral Control:

Up to 10 slow peripheral devices may be driven by a single partition and the IO instruction contains the logical device number, the address of an area of store for the data transfer and a count of the number of bytes to be transferred.

There are four I/O operations:

Read – Transfers up to 10000 bytes of data from the peripheral into store
Write – Transfers up to 10000 bytes of data from main store to the peripheral
Read Control is reserved for loading programs.
Write Control was used by System Ten to generate control codes outside the 64 character set across the MTIOC interfaces and for compatibility, this is retained on System 25. However on System 25 the 8 bit store allows control characters to be sent with a Write Instruction.

4.3 Fast Peripheral Control

For fast peripherals (discs and magnetic tapes) the logical device number is replaced by a Fast Access Channel (FAC) number. Specific numbers have been allocated for different types of fast peripherals as follows:

0 is used for discs.
1 to 4 are used to address up to four tape drives.
5 to 9 are not used.

4.3.1 Disc Control: System 25 Discs are organised as a logical set of sequential 512 byte sectors. The mapping of logical sectors into real track, head, sector numbers is performed by the Disc Adaptor and is dependent on the particular disc characteristics. System Ten discs are organised as logical 100 byte sectors.

The B field in the IO instruction contains a pointer to a 6 byte Control Block which contains the disc drive number (0 to 15), function, count and sector number. System Ten disc transfers were always a single sector of 100 characters. System 25 compatible data disc transfers provide the same operation and five 100 byte blocks are mapped into a single 512 byte sector on the disc and the logical sector number in the Control Block is divided by 5 to give the equivalent 512 byte sector number.

The System 25 disc functions are

Compatible Mode Read or Write – transfers a single 100 byte sector using System Ten Control Block format (maximum sector number = 99,999)

Extended Compatible Mode Read or Write – transfers a single 100 byte sector using System 25 Control Block format (maximum sector number = 9,999,999)

String Read and Write – a string of up to 10,000 bytes starting at the beginning of a sector and extending over several sectors is transferred for one instruction, using System 25 Control Block format (maximum sector number = 999,999)

Other functions are provided to format the disc and read the status of the previous transfer.

4.3.2 Magnetic Tape Control: The use of the IO instruction fields is the same as to the slow peripherals. The Read and Write instructions cause the transfer of a single block of data to or from the tape.

The Read Control instruction is used to access status information from the tape controller and the Write Control instruction is used for control commands for the control of the tape, e.g. Rewind.

Cartridge Magnetic Tape is driven as a 'slow peripheral'.

5 System 25 Software

System 25 software is largely derived from System Ten to allow the large number of application programs developed for that machine to run on System without change.

System 25 can be run with various levels of software support; at the lowest level a program can be loaded directly into a partition and can run and control its peripherals without reference to any supervisory program. The main ICL supporting software is provided in the following three packages, which together give the facilities normally provided by an operating system.

5.1 DMF III (Data Management Facilities)

This is the basic supervisor package and is derived from the System Ten equivalent, DMF II. It consists of the following components:

CSM (Conversational System Manager) which provides facilities for the main-

tenance of the file store and loading programs

LIOCS (Logical Input/Output Control Software) which is a set of housekeeping routines for the logical control of the disc and magnetic tape

Utilities for sorting and editing files

5.2 IAS (Interactive Applications Support)

This provides an environment for transaction processing applications, in which the control of slow peripherals is separated from the application program. It consists of the following components:

Video/Printer Drivers which control up to 10 Videos or printers on a single interface

Data Buffer and Message Handling Routines which are used to pass data between the application programs and the Video/Printer drivers.

5.3 CAM (Communications Access Manager)

This provides a simple macro interface for application programs for the control of communication lines. There is a different CAM package for each communication line protocol. It consists of the following components:

Communication Line Driver which performs the low-level control of the line

Data Buffer and Message Queue handling routines

Macro Interface Routines which convert the macro calls from the application program into messages and data blocks

As was stated in the introduction, the software will be described in more detail in a second paper.

Designing for the X.25 telecommunications standard

K. J. Turner

ICL Information Processing Architecture Division, Kidsgrove, Staffs

Abstract

Packet switching is becoming increasingly important in telecommunications systems in many countries and the 'X.25 Recommendation' of the International Telegraph and Telephone Consultative Committee (CCITT) is now widely accepted as defining the standard for the interface to a public packet-switched network. The paper describes the ICL Communications System Controller, a hardware device with associated software, which has been designed to interface ICL main frame computers to X.25 packet-switched networks. The description is preceded by a short account of X.25 and of ICL's own communication protocol ICLC-03. The essential function of the CSC is to convert between ICLC-03 and other protocols: the conversion between ICLC-03 and X.25 is discussed.

1 Introduction

1.1 X.25

1.1.1 X.25 history: X.25 is probably the most important telecommunications standard to have appeared in recent years. It represents the substantial agreement between communications experts on the interface to Public Data Networks of the packet-switching variety.

Packet-switching began in earnest around 1968 with the opening of networks such as ARPANET (US) and SITA, the international airlines' system. Many countries followed suit in implementing their own packet-switching networks, for example EPSS (UK), RETD (Spain) and TELENET (USA). With great foresight CCITT (International Telegraph and Telephone Consultative Committee) saw the future importance of this developing communications technology and began a programme of work to produce an international specification for interfacing to such networks. Thus was Recommendation X.25[1] born in 1976, to be enhanced within a year by the publication of an alternative link protocol with improved error recovery.

Following the publication of X.25 a considerable amount of activity ensued as older networks were brought into line with the recommendation and new networks

were implemented in direct conformance to it. Typical of these new networks were TRANSPAC (France) and SAPONET (South Africa).

It was inevitable that such widespread and diverse implementations of X.25 would lead to various interpretations of the recommendation and would suggest desirable enhancements to it. CCITT therefore began the revision of X.25 with a view to reconciling the differences in interpretation that had arisen. The result was a new and much larger version of X.25[2] in 1980. At present the conformance of packet-switching networks worldwide to the revised recommendation is patchy, but there is hope of a high degree of commonality by 1982.[3] Recent networks such as PSS (UK) are already largely in agreement with the latest version.

An interesting property of X.25 networks is that it is fairly easy to couple them together, and indeed CCITT has evolved Recommendation X.75[2] for just this purpose. By provision of such 'gateways' between X.25 networks it becomes possible to make calls through the concatenation of several of them. Thus X.25 may lay the foundation for truly international digital data communication.

The following two Sections give a brief overview of X.25. For further background information there are good texts on packet-switching in general[4] and on X.25[5] in particular.

1.1.2 X.25 services and facilities: Packet-switching is a digital communications technique which fragments data into 'packets' of an agreed maximum size and then multiplexes them onto the communications channel with data from other sources. Because the communication channel can carry interleaved packets from many sources its capacity can be utilised very effectively. This is to be contrasted with a circuit-switched connection where plant (lines and switches) will typically be dedicated to that connection. The average utilisation of a communications connection is often low, particularly for interactive traffic, thus leading to inefficient usage of the associated plant. An authority which administers both packet-switched and circuit-switched connections may well be able to offer more competitive rates for a packet-switched connection because the operating capacity may be more nearly reached. The actual choice of connection type must depend, however, on the specific application and on other factors such as the availability of equipment.

Since data from many sources can be multiplexed onto one channel, the packets of each stream need to be distinguished by some kind of address: in X.25 this is termed the Logical Channel Number (LCN). X.25 defines only the interface between the DTE (Data Terminal Equipment, supplied by the manufacturer) and the DCE (Data Circuit-terminating Equipment, usually supplied by the network authority). Many of the parameters of X.25 are therefore local only, that is they apply to the DTE-DCE link. Of course they generally have an end-to-end (DTE to DTE) effect, but usually in a loosely coupled way. The LCN in X.25 is therefore for DTE-DCE addressing only.

To establish a connection from one DTE to another via an X.25 network it is necessary to make what is termed a virtual call, which identifies the destination DTE by means of a unique subscriber number. As part of establishing the call

each DTE-DCE pair will select a logical channel of its link on which to transfer data. The identifier, the Logical Channel Number, of this will generally be different at each end. Completion of the virtual call results in a virtual circuit which persists until the call is terminated (by either party). This kind of temporary association between DTEs is termed a Switched Virtual Circuit (SVC). By way of contrast, a fixed association may also be set up by agreement with the network administration and the other party: this requires no call to set it up or clear to terminate it and is called a Permanent Virtual Circuit (PVC). As shown in Fig. 1, a DTE may have concurrent virtual circuits to several DTEs or even to itself (for test purposes).

X.25 provides a number of options which are available on subscription to the network or on call set-up, as appropriate. They include:

incoming calls barred
outgoing calls barred
reverse charging
call re-direction
charging information
membership of a private group of subscribers
flow control parameters

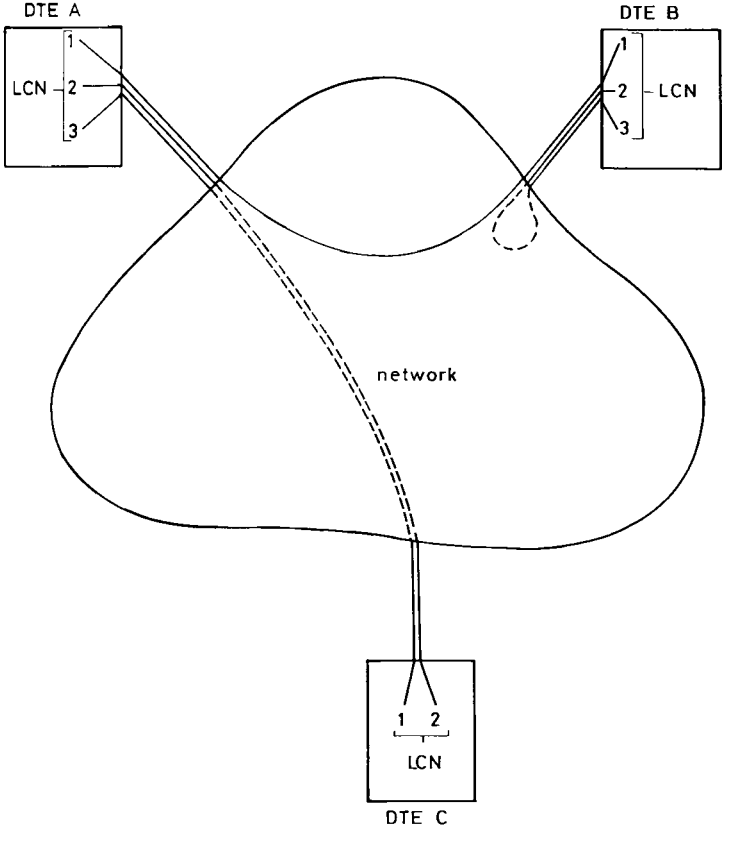


Fig. 1 Examples of Virtual Circuits
—— Permanent Virtual Circuit (PVC)
- - - - Switched Virtual Circuit (SVC)

Most of these have obvious parallels in the telephone network. Only the flow control parameters are unusual in being peculiar to the digital transmission technique used.

The Virtual Circuit service is the only one which is generally available on X.25 networks. A more recent addition to X.25 is the Datagram service which (like Telex) allows the transmission of individually addressed messages. To capitalise on the widespread provision of Virtual Circuits, CCITT has defined a Fast Select facility which in some respects resembles the Datagram Service. A Fast Select call sets up a transient Virtual Circuit, allowing a block of data to be sent with the request. The recipient may either clear the call immediately (optionally supplying a block of data in response) or turn it into a normal virtual call. Thus Fast Select may serve a transaction-oriented application or an enquiry-type of application where the response may be brief or lengthy.

1.1.3 X.25 operation: X.25 is a three-level protocol which conforms to the principle of layer independence. While each layer must know what functions are provided to it by the layer below and what functions are required of it by the layer above, it does not need to have any knowledge of the mechanisms within the adjacent layers by means of which these functions are provided. Such a protocol may be modelled by the familiar 'onion skin' architecture.[6] The three levels of X.25 are:

(i) the Physical (or Transmission) Level
(ii) the Link (or Frame) Level
(iii) the Network (or Packet) Level

In concert they define the procedures across the DTE-DCE interface for establishing connections, transmitting data and breaking connections. The functions of the levels may be briefly summarised as follows:

(i) Physical Level
This concerns the control of the modem (or equivalent) and the transmission of data. Two separate recommendations apply here: X.21[2] (for digital transmission) and X.21 bis[2] (for analogue transmission). X.21 bis (which describes the use of the familiar V-Series modems) is the most commonly found type of interface at present. X.25 connections are full-duplex and so are normally by leased line.

(ii) Link Level
This concerns the control of the DTE-DCE link. Two alternative protocols are used, both of which are variants of HDLC (High-Level Data Link Control): LAP (Link Access Protocol) which is related to the International Standards Organisation (ISO)[7] and the European Computer Manufacturers Association (ECMA)[8] Unbalanced Class, and LAPB (Link Access Protocol Balanced) which is compatible with ISO[9] and ECMA[10] Balanced Class. Both LAP and LAPB offer two-way simultaneous transparent data transfer (with protection against loss, corruption and duplication of data) and flow control facilities. They differ, however, in the way connection, disconnec-

tion and error recovery are handled. LAPB is generally agreed to be superior to LAP and is preferred by CCITT.

(iii) Network Level
This concerns the control of the conversations multiplexed over the DTE-DCE link. The protocol closely resembles that used at Link Level, but offers the important advantages of multiplexing and interconnection of DTEs with differing characteristics (e.g. packet size or throughput). Another important function is the provision of an expedited data route, e.g. for interrupt or break-in.

1.1.4 X.25 Between DTEs: Although X.25 is strictly a DTE-DCE interface standard, it is quite easy to adapt it for point-to-point DTE-DTE links. There are some minor asymmetries in X.25 which can be removed by building sufficient flexibility into a DTE: later Sections mention a few of these. The benefit of X.25 DTE-DTE operation is that the same implementation and protocol can be used for private or public connections.

1.1.5 Related protocols: An X.25 DTE requires a relatively high degree of sophistication that would be expensive to provide in a simple terminal. CCITT identified the need for easy attachment of basic terminals such as teletypewriters to X.25 networks and evolved the interlinked X.3, X.28 and X.29 Recommendations. Like X.25 these have been revised since their initial issue[4].

X.3 defines the behaviour of a Packet Assembler/Disassembler (PAD) which concentrates the traffic from asynchronous terminals onto X.25. X.28 specifies the interface between the PAD and the terminal. X.29 describes the conversation between the PAD and the controlling DTE. X.29 may thus be viewed as a higher – level protocol which is carried over X.25. X.29 does not, however, fit into the more modern communications architectures mentioned in Section 1.2.3 and may therefore be supplanted in the long term by other Virtual Terminal Protocols.

1.2 ICL communications

1.2.1 The ICLC-03 communication protocol: ICL's current communication protocol Full XBM, also known as ICLC-03, is multi-level. The principal components of ICLC-03 are:

(i) Transmission Level
(ii) Link Level
(iii) Group Level
(iv) Access Level

These hierarchically organised levels constitute the means whereby data is transported on behalf of applications or users. There is an obvious similarity to the corresponding functions in X.25.

(i) Transmission level
This concerns the control of the modem (or equivalent) and the transmission of data. Connections may be either half-duplex or full-duplex.

(ii) Link Level
This concerns the control of the link between a primary and one or more secondaries. The protocol is two-way alternate and is derived from the ISO Basic Mode procedures.[11]

(iii) Group Level
This concerns the control of the conversations multiplexed over a primary-secondary link.

(iv) Access Level
This concerns the control of the sequencing and presentation of application or user messages. Unlike the lower three levels, Access Level occurs in several varieties to match the needs of differing devices and functions. Of these, the Device Independent Access Level (DIAL) and the Logical Connection Control Access Level (LCC) particularly concern the subject of this paper.

Like X.25, DIAL offers two-way simultaneous, transparent data transfer and flow control, together with control functions for expedited data and for error recovery. Indeed DIAL and X.25 offer such similar facilities that the mapping between them is almost one-to-one.

ICLC-03 distinguishes clearly between data transfer and connection control functions. LCC provides facilities for mainframe control of connection, disconnection, and error handling. As with DIAL, these map onto X.25 in a natural way.

The close correspondence between ICLC-03 and X.25 makes protocol conversion from one to the other particularly practicable. Some examples of the relationship between the two are given below. The attraction to ICL of ICLC-03 in X.25 protocol conversion is the ease with which large communications systems can be built out of local ICL networks interconnected by public X.25 networks.

ICLC-03	X.25
connect	call
disconnect	clear
reset	reset
access level data packet	data packet sequence
program function	interrupt

For the connection of PAD-concentrated teletypes a similarly convenient mapping of X.29 on to the ICLC-03 Scroll Mode Access Level can be defined.

1.2.2 ICL Communications System Controller CSC: ICL's Communications System Controller hardware (CSC) is designed to front-end the larger ICL 2900 mainframes via a high-speed trunk interface. Communications lines are interfaced by hardware units known as couplers. CSC's primary functions are to act as a concentrator for ICLC-03 communications traffic and to convert other communications traffic to ICLC-03 form. It is in this latter function that the majority of the interest and complexity lies. Currently CSC offers conversion of three classes of protocol:

(i) Asynchronous (teletypewriter)
(ii) Synchronous (video, clustered video)
(iii) Bit-oriented (X.25, X.29)

This paper addresses itself solely to the design of an X.25 protocol converter.

The CSC system software may be broadly divided into Kernel and Communications. A Peripheral sub-system is also used during development. The Kernel offers the usual resource management facilities such as scheduling, buffer allocation, timer control, error handling and hardware interfacing. The Communications subsystem comprises a number of Protocol Converters and the ICLC-03 multiplexer, which may be regarded as a kind of null protocol converter. A significant amount of code is shared between the protocol converters because of the inevitable commonality of functions to be performed. The general structure of CSC and its relationship to the mainframe are illustrated in Fig. 2.

After the next Section giving a short note on the ICL general information processing architecture IPA, the rest of the paper is concerned with CSC. The three main Sections deal with the following aspects:

Section 2- Functionality. The services provided by CSC.
Section 3- Architecture. The basic design of CSC and an indication of the mechanisms by means of which these services are provided. In particular, three important software components - Driver, Connector and Multiplexer - are described.
Section 4. Resource Management. Scheduling, flow control, handling of errors and collection of statistical information are described.

The final Section deals briefly with the testing of this equipment.

1.2.3 ICL Information Processing Architecture IPA: Considerable interest has been shown in recent years in the development of Open Systems Interconnection (OSI).[6,12] The concept which underlies OSI is that any host or terminal in a community should be able to interwork with any other. As a step towards this goal ISO have evolved a Reference Model[13] which defines an appropriate system architecture. The functions of this model are separate into seven layers, each of

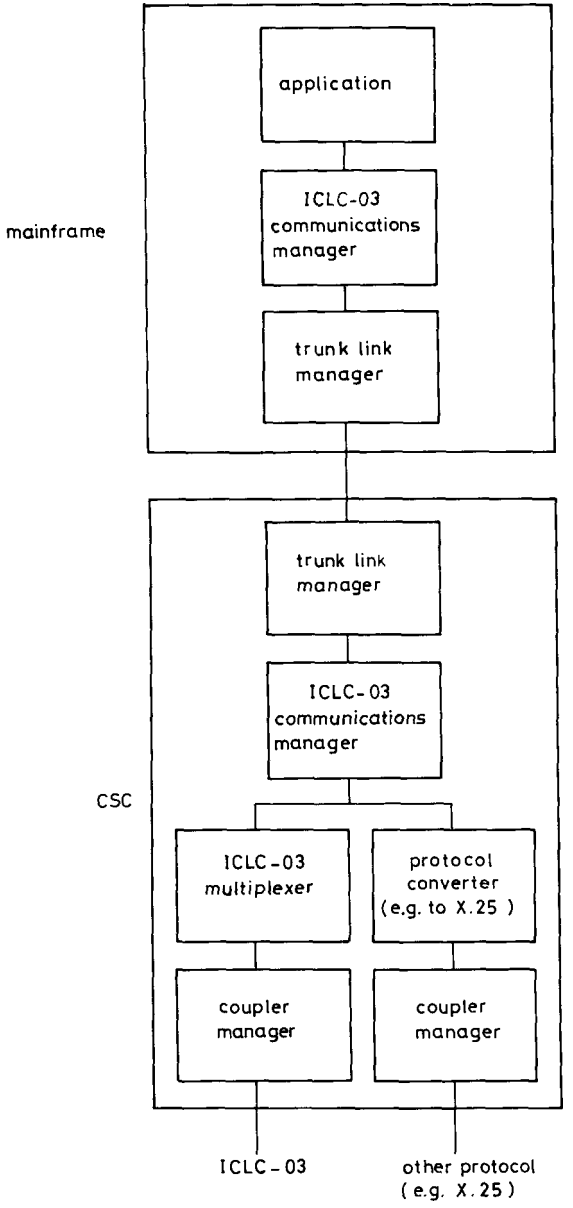


Fig. 2 CSC structure and relationship to mainframe

which builds progressively on the layers below it. The principal layers are:

Information Processing Functions	7 Application Layer	(user programs/facilities)
	6 Presentation Layer	(device characteristics)
	5 Session Layer	(synchronisation)
Telecommunications Functions	4 Transport Layer	(end-to-end-connection)
	3 Network Layer	(routing)
	2 Data Link Layer	(data transfer)
	1 Physical Layer	(electrical)

ICL has adopted the OSI model as the basis of its Information Processing Architecture (IPA).[14] The particular relevance of X.25 to IPA is that it can serve as the major part of the lower three layers of the model.

2 Functionality

2.1 Split of functions

As explained in the paper by Kemp and Reynolds,[14] there are many possible ways of partitioning the layers of the IPA model between different functional units. For protocol conversion of X.25 the most natural choice was to separate the lower three layers and implement them in CSC. The networking capability is therefore vested in CSC and the end-to-end control is handled by the mainframe.

The other split of function between the mainframe and CSC is that of control as opposed to data transfer. The general philosophy of ICLC-03 is that the mainframe is responsible for overall control. It is therefore the mainframe which initiates conections, breaks them and decides how to recover from serious errors, although it is CSC that carries out the detail of these operations.

2.2 Services

As well as standard X.25, CSC also implements the ICL DTE-DTE variant for point-to-point operation.

At the Physical Level X.25 has a choice of connection service: X.21 or X.21 bis. CSC supports only the latter.

At the Link Level X.25 has the option of operating with either of the access protocols LAP, LAPB referred to in Section 1.1.3(ii). Both of these are implemented in CSC because the older networks have not yet converted from LAP to LAPB.

It was explained in Section 1.1.2 that X.25 offers two main types of service: Virtual Circuits and Datagrams. It was also noted that Virtual Circuits, which are very similar to physical circuits, fall into two classes: Switched Virtual Circuits (SVCs) which are analogous to Public Switched Telephone Network connections,

and Permanent Virtual Circuits (PVCs) which are analogous to leased-line connections. The concept of a Datagram is rather foreign to many communications protocols, which usually follow the pattern of connect/exchange transactions/disconnect. In particular, the mapping of datagrams on to ICLC-03 would be fairly difficult. It is also unlikely that datagrams will be widely available on X.25 networks in the near future.

CSC therefore supports the Virtual Circuit service only. Besides the basic SVC/PVC Split mentioned above, Switched Virtual Circuits are further categorised as Inward (ISVC), Bothway (BSVC) or Outward (OSVC). These labels refer to which end originates the call, not to which end originates the data since a virtual circuit is intrinsically symmetrical once it is set up. Note that it is really the logical channels of the DTE - DCE interface, rather than the virtual circuits, which have a call direction.

2.3 Facilities

Facilities in the X.25 sense refer to the options available with a particular service. In the case of CSC they define the properties of the virtual circuits. When a Switched Virtual Circuit (SVC) is set up the required facilities are specified in the call request or are given default values. X.25 allows facilities, particularly those related to flow control, to be negotiated. The recipient of the call may either reject the call if the requested facilities are unsuitable or accept the call with modified facilities. This dynamic set-up of facilities is not available on a Permanent Virtual Circuit; instead, the facilities must be agreed at subscription time with the network authority and sometimes also with the other party.

The choice and negotiation of call facilities is not really an appropriate matter for a front-end processor like CSC. For example, the decision to accept a reverse charge call may require reference to a file of accredited users. CSC therefore handles all facilities transparently and leaves call analysis or synthesis to the mainframe. Although the same level of sophistication is not needed at the Link or Physical Levels their connection is handled in the same way for consistency.

A wide range of facilities is available on virtual circuits. The ones permitted by CSC are as follows, although it should be noted that the support of a number of them is dependent on the host operating system:

fast select
reverse charging
call re-direction
call charge notification
membership of a private group of subscribers
variable choice of flow control parameters
negotiable choice of flow control parameters
sub-addressing
packet re-transmission

The facility names are fairly self-explanatory, but a few require further description. Sub-addressing allows the network subscriber address to be extended by further

information to determine the process or function required of the called DTE. CSC does not use sub-addressing since the call request has fields which allow this requirement to be specified more comprehensively. Packet re-transmission refers to the ability of the DTE to retransmit an unacknowledged packet on request. This is not needed for interfacing to an X.25 network but may be required for DTE-DTE operation, where symmetry is necessary.

It is difficult to design a single DTE implementation which will work against all or even most X.25 networks because the facilities offered, and sometimes their encodings, vary widely.[15] The general approach with CSC was to support the common features and to tolerate variations in the protocol where these were known. Detailed study of individual network specifications was often necessary to determine incompatibilities.

2.4 Parameters

The operation of each level of X.25 is characterised by a large number of parameters. At the Physical Level they specify the modem and line characteristics. At the Link Level they specify the frame size, time-outs, retry limits, etc. At the Network Level they specify the packet size, throughput requirement, circuit type, etc. Although some of these parameters (such as the Link Level addresses) are fixed by convention in X.25, CSC permits their values to be varied to maximise the range of applications in which CSC may be used. Thus DTE-DTE X.25 operation at Link Level is simply achieved by giving one end the X.25 DTE address and the other end the X.25 DCE address.

ICL's 2900 operating systems have the concept of a catalogue which defines the potential hardware configuration. Configuration parameters are held centrally in this catalogue and are distributed when connections are set up. CSC therefore obtains its X.25 parameters dynamically as required. One consequence of this is that CSC is unaware which logical channels correspond to each type of Virtual Circuit. This is convenient in DTE-DTE X.25 operation since what is an Inward Switched Virtual Circuit at one end is an Outward Switched Virtual circuit at the other.

3 Architecture

3.1 Protocol conversion

In general terms, protocol conversion requires the abstraction of the essential concepts of a protocol, such as 'message', 'acknowledgment' or 'reset' and the definition of their mapping onto another protocol. A protocol converter takes responsibility for operation of the outboard protocol and wherever possible hides the details of this from the mainframe. For example, a protocol converter performs its own flow control and undertakes autonomous error recovery, within prescribed limits. Where the two protocols to be mapped are hierarchically layered it may be possible to set up a good correspondence between the levels of each protocol. It might then seem that conversion could simply be carried out on a level-by-level basis. In practice this is not normally feasible since although the functions of two

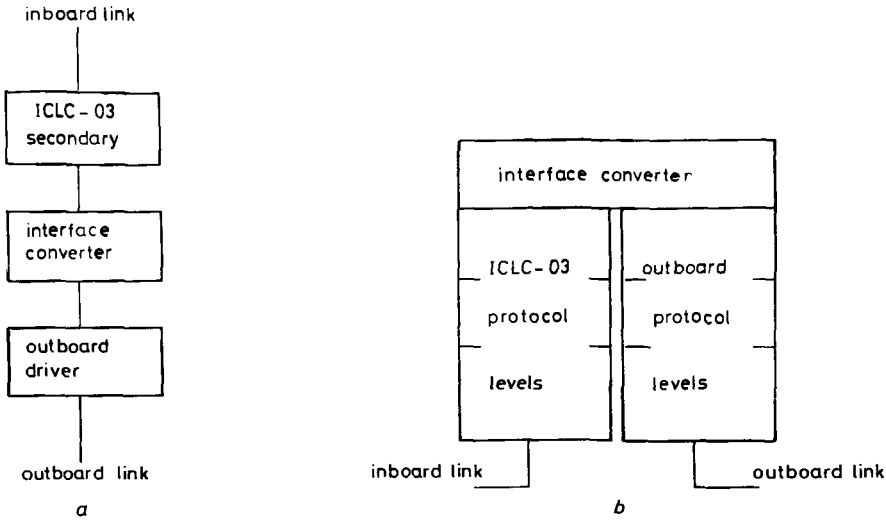


Fig. 3 Protocol Converter mode I
(*a*) Linear
(*b*) Layered

levels may correspond fairly well the means of achieving them usually do not. This is the case with protocol conversion between ICLC-03 and X.25.

Protocol conversion is generally quite complex, but is made easier in CSC because of its specially designed architecture. Fig. 3 shows two representations of the abstract protocol converter model developed for CSC. The linear representation is useful for explaining the data flow through CSC whereas the layered representation is useful for showing the correspondence between the two protocols. The ICLC-03 Secondary component is what the mainframe sees; the task of the protocol converter is to make this reflect the outside world as faithfully as possible. The Outboard Driver software is responsible for the levels of the target protocol. The Interface Converter is at first glance redundant, and indeed is a null function in some of the CSC protocol converters. However, it serves the useful purpose of allowing different varieties of ICLC-03 Secondary and Outboard Driver to be coupled. For example, the Scroll Mode Access Level may be coupled to Asynchronous Driver for local teletypewriters or to the X.25 Driver for remote teletypewriters concentrated by a Packet Assembler/Disassembler (PAD). Conversely the X.25 Driver may be coupled to the Device Independent Access Level for X.25 conversion (see Section 1.2.1(iv)) or to the Scroll Mode Access Level for X.29 conversion.

The structure of the X.25 Protocol Convertor is shown in Fig. 4 in both forms. Also shown is the relationship of the connection control function to the levels of the model. Connection control appears as a separate Group and Access Level in parallel with those of the data route.

3.2 Relationship between levels

Each connection at each level is represented in CSC by a *program instance*, that

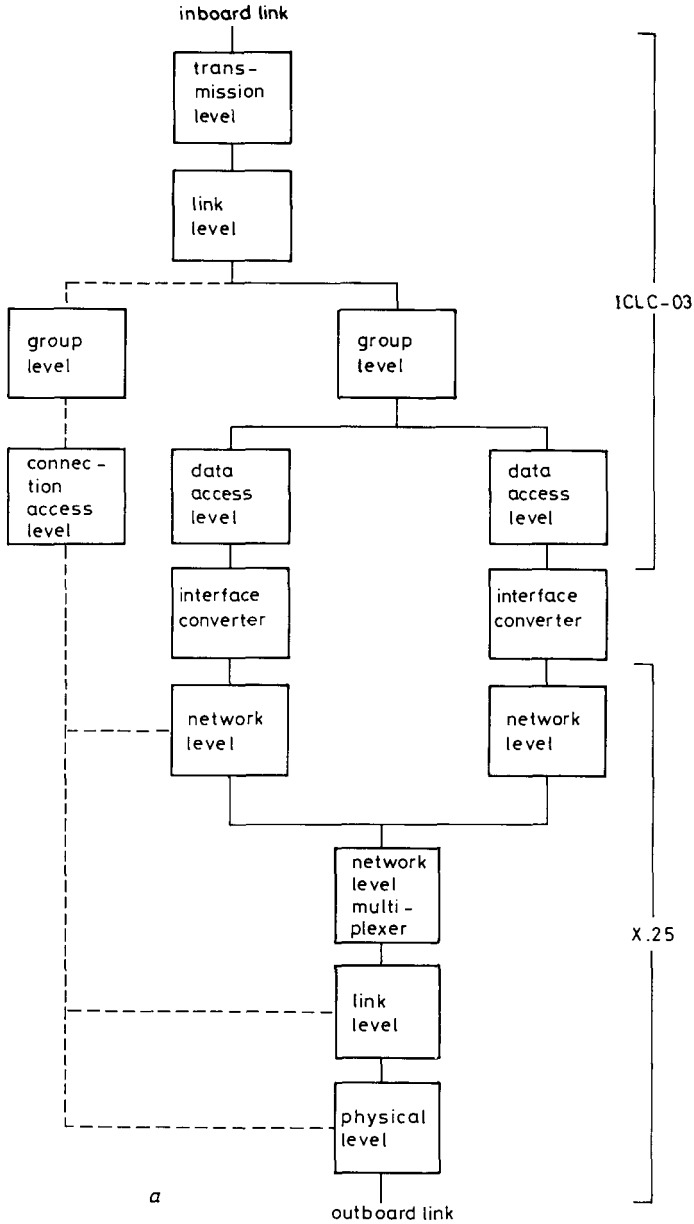
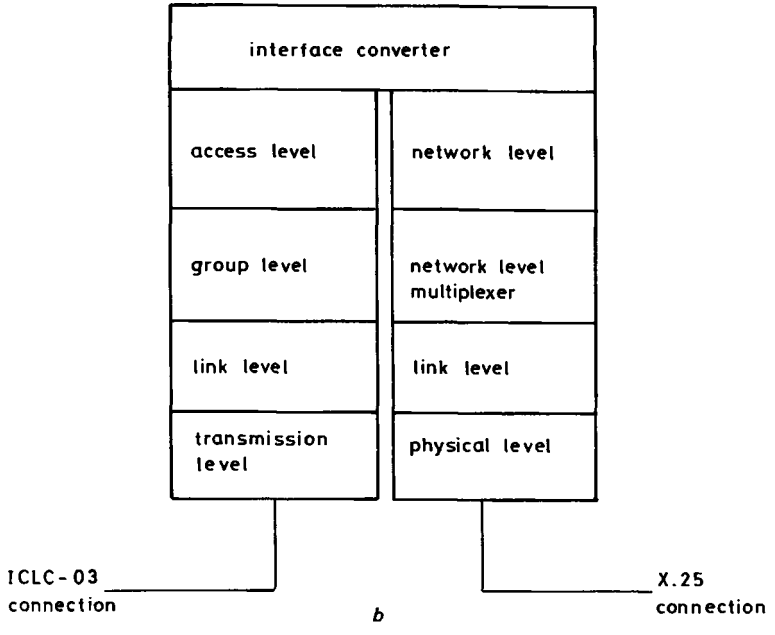


Fig. 4 ICLC-03/X.25 Protocol Converter
(*a*) Linear
(*b*) Layered
—— data flow
– – – control flow



b

is, a piece of shared code together with the data structures particular to that connection. In X.25 the relationship between Physical and Link Level instances is one-to-one and that between Link and Network Level instances is one-to-many. However, for reasons of throughput and resilience it is highly desirable to have several physical connections to the network. At some point this multiplicity of connections must be brought together to allow alternative routing of messages. This gives rise to three variations:

(i) Multi-line, where one Link Level is carried over several physical connections.
(ii) Multi-link, where each Link Level is carried over just one physical connection but the Network Level is supported by several Link Levels.
(iii) Multi-circuit, where the mutiplexing is carried out at a higher level than X.25 (for example in the Transport Layer).

The first two possibilities require an intermediate level of protocol between the Physical/Link or Link/Network Levels. The third possibility has the attraction of requiring no change to X.25. It would be feasible to employ multi-line or multi-link operation at the same time as multi-circuit operation since the former provides resilience to line failures and the latter provides resilience to network failures. Since CSC was designed with a view to embracing the structure of X.25 variants it has the capability of representing the relationship between two levels as one-to-one, one-to-many, many-to-one or many-to-many. This flexibility would also allow the CSC X.25 implementation to be used in other contexts, for example for pure HDLC: in

this case there would be no Network Level in CSC. Some examples of the structures that might be built in this way are shown in Fig. 5. Although CSC does not have the protocol elements to do other than X.25, it was felt to be important to devise the correct architecture to allow them to be slotted in if required. CSC models the relationship between levels by chained entries in Physical, Link and Network Level tables. These are tied together under the overall ownership of the Stream table, a stream being the logical level of multiplexing across the CSC-mainframe interface. Each table entry describes the address and status of an instance and points to its data space. The instance's data space contains its control block, parameters and work variables.

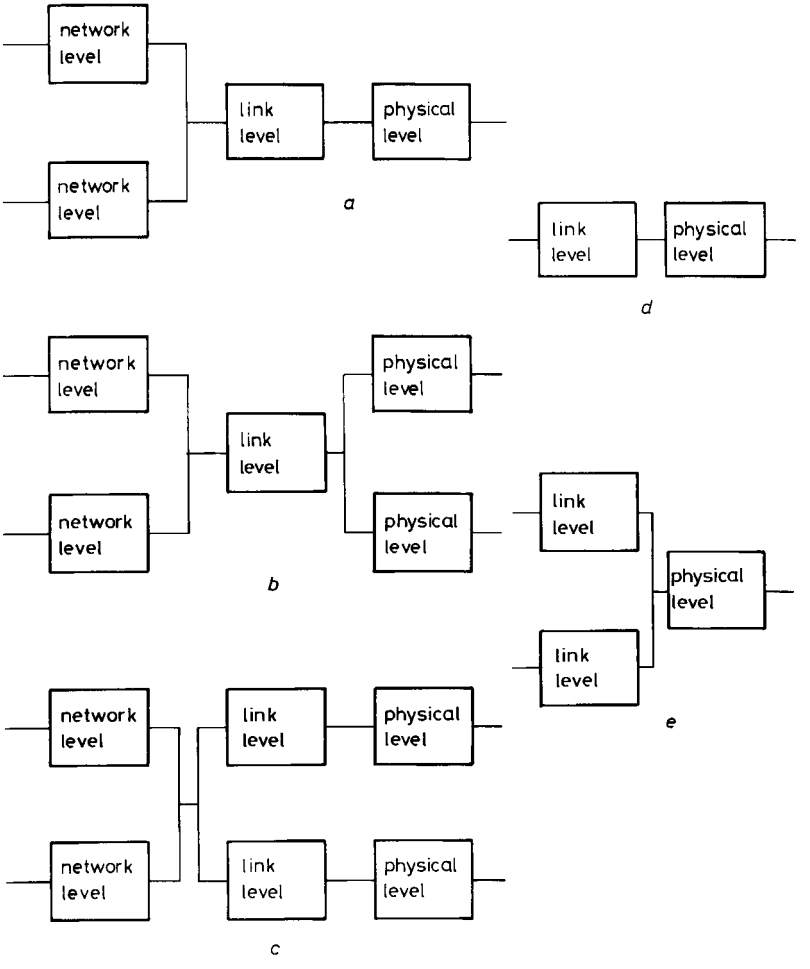


Fig. 5 Structures that can be represented in CSC
(*a*) Standard X.25
(*b*) Multi-line X.25
(*c*) Multi-link X.25
(*d*) Standard HDLC
(*e*) Multi-point HDLC

instance and points to its data space. The instance's data space contains its control block, parameters and work variables.

As explained in Section 3.3, every effort was made to make the various levels conform to the same pattern. Thus the table entries and their corresponding data spaces have the same format. This means that the three-level structure adopted for X.25 could in principle be extended to any number of levels although in practice this has not been necessary.

3.3 Level Model

3.3.1 General structure: The natural architecture for a hierarchically structured protocol such as X.25 is one in which the individual levels are rigidly isolated. This preserves the independence of the levels and results in a clean functional split when it comes to implementation and testing. The principle has been carried even further in the design of the CSC X.25 driver in that the individual levels have been made as far as practicable to follow the same model. This approach has several advantages:

(i) Understanding and maintaining an admittedly complex piece of software is made easier.
(ii) Sharing of code between levels (in particular, the Link and Network Levels) is practicable.
(iii) Design and documentation effort can be reduced owing to the commonality of interfaces.
(iv) A single test-bed can be used to develop all of the levels.

Viewed externally a level has two types of interfaces: data transfer and connection control. The basic commands defined on these interfaces are as follows:

data transfer	LOAD PARAMETERS	– initialise level instance
	READ	– input a message
	WRITE	– output a message
	QUIESCE	– abort outstanding transfer requests
connection control	CONNECT	– create level instance
	DISCONNECT	– delete level instance
	ABANDON	– abort outstanding connection requests

A number of variations on these are defined in the command parameters. For example, READ and WRITE are qualified 'control' (e.g. reset, interrupt) or 'data'

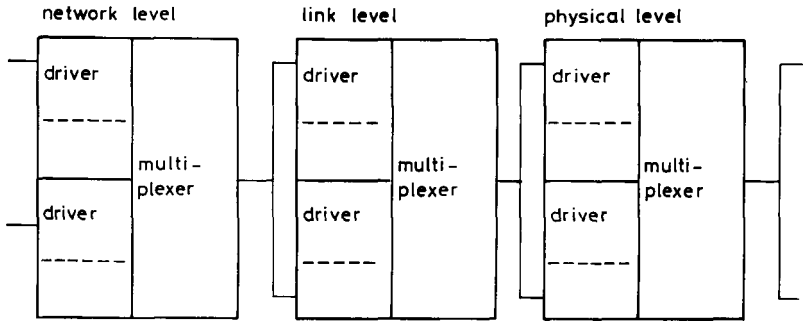


Fig. 6 Relationship between data transfer instances

(pure data). Both the data transfer and connection control interfaces use the same underlying mechanisms although there are a few differences: for example, the data transfer interface may be operated in a two-way simultaneous fashion as well as two-way alternate.

3.3.2 Data transfer: The Network Level is in some senses the most general of the three levels for it contains both data transfer and multiplexing sub-levels. Section 3.2 describes the concept of multi-line and multi-link operation, where it may be necessary for a level to drive several lower-level instances. This may also happen at the Physical Level if the communications coupler is multi-channel. The general model of X.25 data transfer in CSC embraces these possibilities as shown in Fig. 6.

Each Driver in the model handles the data transfer of one level instance (line, link, or virtual circuit). The Multiplexers at each level serve two functions: they multiplex the traffic at one level onto the lower level and they isolate the Drivers from each other. If CSC were to implement multi-line or multi-link procedures the associated protocol would be handled in the Multiplexers. In this sense the Multiplexers may be viewed as intermediate sub-levels of the three main levels.

3.3.3 Connection control: Connection control is exercised by a separate hierarchy of levels which parallels the data hierarchy. There is a deeper point here which is not fully exploited in CSC, namely that the function of a level can be viewed from many aspects: data transfer, connection control, error management, testing, etc. Thus many hierarchies co-exist as different facets of the same architecture. Each CSC level has a Connecter function associated with it. In the interest of commonality the Connecters all conform to the same internal interface standard.

Connection of a level instance requires the higher level to be notified that a new instance is available for use. Conversely, disconnection requires the higher level to be told to delete the instance from its scheduling queues. To avoid conflict over the use of the instance, a control function is provided which allows higher level data transfer to be disabled pending re-connection or disconnection.

Summarising these points, connection is exercised via the Connecter (for

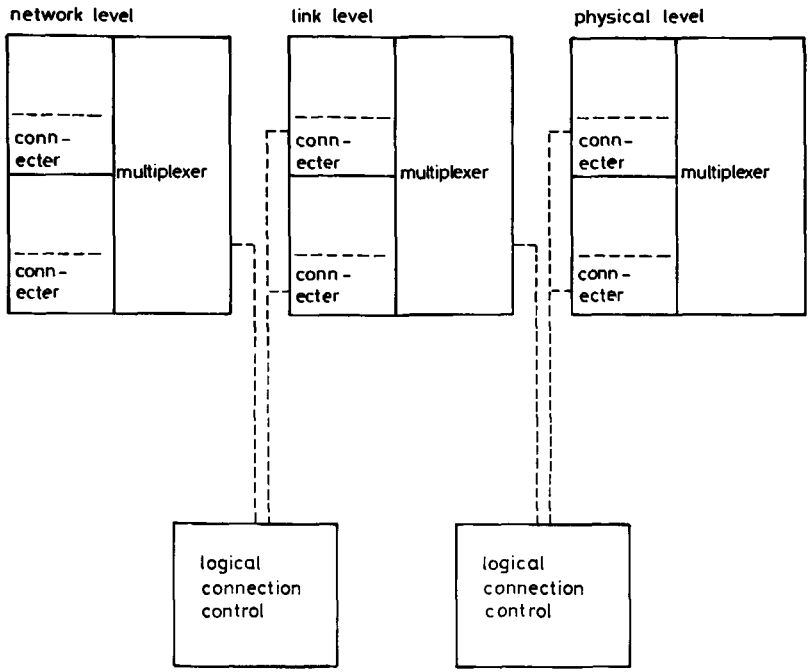


Fig. 7 Relationship between connection control instances

connection of a level instance) and via the Multiplexer (for use of a lower level). This is illustrated in Fig. 7.

3.3.4 Composite model: Previous Sections have introduced the concepts of the Driver, Connecter and Multiplexer. An instance of the Driver and Connecter exists per line, link or virtual circuit, but the Multiplexer is single-instance per level. The functions of these three components are as follows:

Driver
- interfacing to higher level
- message transfer
- error detection and recovery
- timer control

Connecter
- interfacing to connection control
- overall control of instance
- error reporting

Multiplexer
- interfacing to lower level
- multiplexing
- scheduling

The relationship between the Connecter and the Driver is symbiotic. As well as performing message transfer for the Connecter, the Driver also looks after other functions such as time-out and error recovery. Only if the Driver detects a serious or persistent error does it inform the Connecter. In error conditions or during (dis)connection the Connecter can disable the Driver, preventing input or output. Errors are notified to the mainframe, where the decision to recover or abandon the connection is made. If the Connecter is told to continue operation it resets the connection and enables the Driver.

The Multiplexer performs input and output on behalf of the Drivers, thereby isolating them from the lower level. One aspect of this is that the Multiplexer takes responsibility for the message header (address and control fields). Supervision of the address field is clearly appropriate since it is intrinsic to the multiplexing function. The Multiplexer maintains a list of connected instances at its level so that it can route incoming messages on the basis of their address. Supervision of the control field is less obviously appropriate since this concerns Driver or Connecter operation. However, CSC can optimise on buffer usage when dealing with supervisory messages (which are quite common). Since these consist of the message header only it is not necessary to allocate a buffer to them. The Multiplexer therefore extracts or inserts the control field along with the address field, but allows the Driver or Connecter direct and transparent use of it.

For simplicity all Connecter messages are routed through the Driver so that the Multiplexer has to schedule transfers on behalf of Drivers only. There is a very

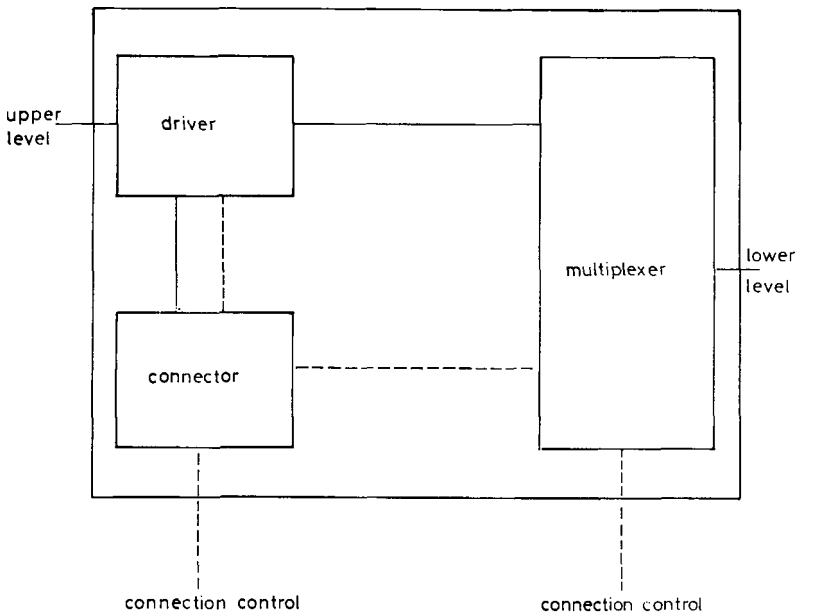


Fig. 8 Composite model of an X.25 level
—— data flow
---- control flow

simple and convenient division of message types between the Driver and the Connecter: the former receives numbered messages (data, acknowledgements, etc.) and the latter receives unnumbered ones (connect, reset, disconnect, etc.) Since the Multiplexer has to handle the control field anyway, it carries out this numbered/unnumbered categorisation although it does not perform the Driver/Connecter split. Because of the structure of X.25 control fields it is possible for the Multiplexer to categorise message types without being aware of their meaning.

Some X.25 functions require co-ordinated control of all the instances at a level: specifically, this is needed on disconnection of a level (which leaves upper levels unsupported) and on Network Level restart (which re-initialises all circuits). The Multiplexer takes responsibility for this overall control of a level, making use of the list of connected instances which it maintains for multiplexing purposes.

The above observations on the inter-relationship between the Driver, Connecter and Multiplexer are brought together in the composite model in Fig. 8.

4 Resource Management

4.1 Scheduling

Although the Kernel software in the CSC (see Section 1.2.2) provides overall system scheduling, the X.25 subsystem is responsible for allocating processing to individual instances at each level. To prevent excessive path lengths through this software each level is scheduled separately. There is normally just one instance at the Physical and Link Levels so scheduling is unnecessary. However, consistent with the principle of making the levels resemble each other a general output scheduling algorithm was designed.

This algorithm assigns to each instance a priority which is defined such that its throughput is twice that of an instance at the next lowest priority. Certain complexities arise in the design: for example, it is desirable to service higher priority requests which occur during processing of a lower priority instance, yet higher priorities must not be allowed to lock out lower ones for long periods. This is achieved by allocating a processing quota to each priority. The quota expresses the relative throughput required and is dynamically calculated from the number of instances and average message length for that priority level.

Those familiar with X.25 will have realised that the scheduling priority closely resembles the Throughput Class defined for virtual circuits, and this indeed was the justification for an algorithm of this form. However, it was reckoned that the cost of running such an algorithm would be a significant contribution to the CSC processing load. A simple single-priority scheduler was therefore implemented. Although this means that a mixture of bulk and interactive X.25 traffic may not be optimally handled it does result in a compact and efficient design.

Two considerations in the design of the output scheduler were how to determine which instances required service and whether to lock on to active instances. The

scheduling system adopted operates 'on demand', that is, Drivers requiring service (i.e. output) notify the Multiplexer and have their requests queued by it. The alternative procedure of having the Multiplexer scan the Drivers for traffic was judged to be less efficient as the Drivers are usually inactive. Since a Driver is allowed to have only one service request outstanding (but may have others pending) the question arose of whether to remain with a Driver until all its requests were satisfied. If a message is preserved until all its component frames or packets have been acknowledged, locking on to a Driver could be beneficial as far as buffering is concerned. However, CSC can delete buffers as soon as acknowledgements are received. It was therefore decided to service requests one at a time in strict rotation of Drivers, thereby precluding heavily utilised instances impeding traffic on others.

Once the Multiplexer has selected a Driver to service, the allocation of a lower level instance to carry out the request is quite straightforward. The Multiplexer maintains a queue of lower-level instances that are available for output and simply selects the first of these. If several alternative physical connections were available to an X.25 network this algorithm would automatically ensure that output traffic was allocated to them in proportion to their capacity. (The physical connections need not be all of the same speed).

As far as input scheduling is concerned, the chronological order of message arrivals determines the processing sequence. The only point worth mentioning is that the method used in CSC for communicating between levels is more efficient when input responses are passed in bursts. The Multiplexer therefore does not lock on to a lower-level instance which has input, but takes input messages in strict rotation from all the instances.

4.2 Buffering

X.25 is defined such that the units of Physical, Link and Network Level transmission are the same. It is therefore feasible to share the buffers of a message between all levels. There are two subtle pitfalls in doing this, however. One is that an acknowledgment for message buffers may be received at a higher level before lower level acknowledgment has yet been received. Although the provisions of X.25 exclude this possibility, the DTE must protect itself against possible errors since the consequence would be continued use of buffers which have been freed elsewhere in the system. The other pitfall is encountered if a request to retransmit parts of a message is received at a higher level before lower levels have yet finished with the associated buffers. It may then happen that a level is required to have the same buffers queued for transmission more than once. This is not possible in X.25 proper, but may happen in DTE-DTE X.25 operation.

CSC implements a general solution to both the above problems. Each level is allowed to queue the buffers associated with a message once. But each level (and sub-level, to allow for the Multiplexers) must mark the buffers as being in use. Only the last user of the buffers may free them. A request to re-transmit a buffer which is still queued at a lower level is suspended until the lower level has finished. Clearly this approach may be applied to any number of levels.

The question arises of whether the same message buffers can be shared between both the ICLC-03 secondary and X.25 levels. Unfortunately this is not possible because both halves of the protocol convertor operate independently. Particular difficulties in sharing buffers would occur in the event of a reset on one side or the other. CSC therefore copies the message in the Interface Converter. Although this imposes a performance penalty there are several advantages in doing so:

(i) Message fragmentation can easily be changed (for example, the ICLC-03 Link Level packet size is fixed but a variety of X.25 packet sizes can be catered for).

(ii) Character conversion or checking can be performed simultaneously. This is not necessary for pure X.25 interfacing but is essential for X.29 where character parity, for example, must be generated or checked.

(iii) Autonomous operation of the inboard and outboard protocols can be preserved.

4.3 Flow Control

X.25 offers a variety of facilities for data flow control. Indeed these overlap sufficiently that it is not clear what subset to choose. Although the message size and throughput parameters have a bearing on the flow control, the primary mechanism is a 'window' which limits the number of messages that may be outstanding, awaiting acknowledgment. There are several ways to regulate the flow of input data:

(i) Acknowledgments to input can be delayed. This is possible but not recommended at Link Level, but is permissible at Network Level. The disadvantage of this technique is that sufficient buffering must be permanently allocated to accommodate the window. The effect on the remote station is also undesirable in that it does not know to cease transmission until the window is full, so its own buffering is increased.

(ii) The remote station can be told to suspend transmission. This is achieved at Link or Network Level by a special supervisory message, but it may not prevent the window becoming full since there may be messages already in the pipeline. However, the remote station is advised of the blockage as soon as possible.

(iii) Input data which cannot be buffered can be discarded and a request for re-transmission issued later. This is permissible at Link Level but is not always permissible at Network Level, since only some X.25 networks offer packet re-transmission. The advantage of this technique is that buffers do not need to be locked down for each circuit in case the window becomes full. The disadvantages are the need for re-transmission, and the limitation to those networks which support packet re-transmission.

CSC uses technique (ii) for its flow control, but can also use technique (iii) with suitable networks. This has the advantage that the same algorithm could be used at

Link and Network Level to maximise commonality. However, as already observed in Section 4.2 the X.25 levels share the buffers of a message. There is therefore little point in controlling the input of the same buffers at more than one level. In order to preserve the independence of the access level conversations CSC operates its flow control at Network Level only.

4.4 Error control

Although CSC has autonomy in managing its X.25 operations, it is ultimately subject to mainframe control. CSC therefore notifies the mainframe of any change in connection status: successful (dis)connection, unsuccessful (dis)connection, unsolicited (dis)connection, or unrecoverable error. The reason for a failure is specifically identified and diagnostic information (e.g. the message which caused the error) is supplied.

Many error conditions are possible in X.25. A number of these are fairly harmless and can safely be ignored by a DTE. CSC adopts a defensive approach to error detection and in general checks for only these errors which would cause subsequent (and often indeterminate) failure if untrapped.

X.25 failures are often context-dependent: to resolve them requires knowledge of the protocol exchanges prior to the error. A convenient feature of the inter-level interface mechanism in CSC is that a trace of all messages is kept. The progress of a message through the levels or the history of the messages at a level may therefore be determined.

4.5 Statistics

There are many reasons for maintaining statistics about X.25: for example, performance assessment, reliability monitoring and network accounting. Because of the cost of keeping statistics CSC maintains only a rudimentary set, which nonetheless enables a variety of interesting information to be derived. An overall system picture can be obtained by combining the mainframe and CSC statistics.

The same set of statistics is computed by CSC at each X.25 level although the meaning of terms like 'supervisory message' and 'failure' obviously differs from level to level:

(i) Total number of data messages
(ii) Total number of supervisory messages
(iii) Total number of failures (recoverable and unrecoverable)
(iv) Totals peculiar to internal CSC operation

From these statistics it is possible to calculate factors like the throughput, the line error rate and the effectiveness of the X.25 double-numbering (which allows acknowledgments to be carried in data messages). CSC tuning information can also be obtained.

5 Testing

5.1 Development aids

Because of the complexity of the CSC X.25 developments it was essential to provide a graded series of test environments, leading up to the reference test described in the next Section:

(i) Isolated testing of levels
(ii) Hierarchical testing of levels
(iii) Testing against a protocol simulator
(iv) Testing in loop-back mode
(v) Testing with point-to-point connection

To facilitate the off-line testing of levels two test-beds were developed: a Higher-Level Simulator and a Lower-Level Simulator, which simulate the interfaces to a level. One consequence of designing all levels to a common interface standard was that the same test-beds could be used to test the Physical, Link or Network Level. Furthermore the test-beds could be operated independently at any level, so that a level could be checked in isolation or a set of levels could be checked in conjunction. It was therefore possible to develop all of the X.25 code in parallel, not in series as might have been expected.

A variety of communications monitoring equipment is now available with an X.25 capability. The actual facilities offered vary widely between manufacturers, in some cases being little more than HDLC and in others being full network simulation. Two testers were used during CSC development: one with basic X.25 features but flexible enough to generate any protocol sequence, and one with complete simulation of specific networks. The use of the basic X.25 tester was justified by the fact that there are many conditions that do not usually arise in normal transfer but which must be checked.

Loop-back and point-to-point testing are virtually the same, but the latter gives more confidence in that two systems are involved. As mentioned in previous Sections the asymmetries of X.25 can easily be overcome, which makes self-testing quite feasible.

5.2 Reference tests

Ultimately the X.25 DTE must be connected to a real network. Although network administrations differ in their requirements the manufacturer must usually demonstrate conformance to electrical safety standards and to X.25 protocol standards. The network administration carries out an initial investigation of this and grants temporary 'permission to connect' if the results are satisfactory. Full permission is not granted until the DTE's implementation of the protocol has been confirmed by operation on the network for a trial period. It is very useful if a DTE can be checked in advance against a reference tester for the network. Some commercially available communications monitors have been approved for this purpose, and as already mentioned one was used during CSC development.

6 Conclusions

It will be apparent from this paper that many interlocking design decisions have to be made in the implementation of an X.25 DTE. This is aggravated by the nature of the X.25 Recommendation itself, which is almost exclusively DCE-oriented. Experience on the CSC has shown that a properly designed arthitecture considerably eases the implementation of a complex protocol like X.25. Careful isolation of layers and functions allows the task to be partitioned into manageable units and facilitates testing and maintenance. Although the CSC design is only one of several feasible alternatives, it is hoped that its principles may prove of interest to others working in this challenging and important field.

Acknowledgments

Several people within ICL and the CSC project have helped to mould the design presented in this paper. Particular mention must be made of R.J.Gillman who collaborated on the overall design and who developed network-independent standards for ICL implementation of X.25. The architectural model stems from the pioneering work of D.J.Ackerman. Thanks are also due to Carol Thorley for typing the manuscript.

References

1 CCITT: X-Series (Public Data Network) Recommendations, Orange Book Vol. VIII.2. International Telecommunications Union, Geneva, 1977.

2 CCITT: X-Series (Public Data Network) Recommendations, Yellow Book Vols. VIII.2 and VIII.3. International Telecommunications Union, Geneva, 1981.

3 RYBCZYNSKI, A.M. and PALFRAMAN, J.D.: A common X.25 Interface to Public Data Networks *Computer Networks,* 1980, **4**, 97-110.

4 DAVIES, D.W., BARBER, D.L.A., PRICE, W.L. and SOLOMONIDES, C.M.: *Computer Networks and their protocols.* Wiley, New York, 1979.

5 SLOMAN, M.S.: X.25 Explained. *Computer Communications, 1978,* **1**(6), 310-326.

6 HOULDSWORTH,J.: Standards for open network architecture. *ICL Tech. J.,* **1**(1), 1978, 50-65.

7 ISO: International Standard 6159 (HDLC Unbalanced Class of Procedures). International Organisation for Standardisation, Geneva, 1980.

8 ECMA: Standard 60 (HDLC Unbalanced Class of Procedure). European Computer Manufacturers Association, Geneva, 1979.

9 ISO: Draft International Standard 6256 (HDLC Balanced Class of Procedures). International Organisation for Standardisation, Geneva, 1978.

10 ECMA : Standard 61 (HDLC Balanced Class of Procedure). European Computer Manufacturers Association, Geneva, 1979.

11 ISO: International Standard 1745 (Basic Mode Control Procedures). International Organisation for Standardisation, Geneva, 1971.

12 BRENNER, J.B.: Using Open System Interconnection standards. *ICL Tech. J.,* 1980, 2(1), 106-116.

13 ISO: Draft Proposal 7498 (Open Systems Interconnection Basic Reference Model). International Organisation for Standardisation, Geneva, 1980.

14 KEMP, J. and REYNOLDS, R.: The ICL information processing architecture IPA. *ICL Tech. J.,* 1980, 2(2), 119-131.

15 HESS, M.L., BRETHES, M. and SAITO, A.: A comparison of four X.25 Public Network Interfaces. Proceedings of the International Conference on Communications, Boston, 1979, 38.6.1-38.6.8.

Viewdata and the ICL Bulletin System

D.R.Olivey
Baric Computing Services Ltd., Feltham, UK
R.Sugden
ICL, Feltham, UK

Abstract

Viewdata is one of the newest areas of interest in computers. The paper describes the design and implementation problems encountered in producing ME29 BULLETIN, ICL's first Viewdata product.

1 Introduction

Viewdata, or VIDEOTEX as it is now becoming called, is as yet a loosely defined area of computer systems, but at its broadest it is concerned with making online computer systems available to a mass audience. It attempts to achieve this by building on the components of current TP technology, standardising and simplifying wherever possible.

Its current implementations, centred around PRESTEL,* are mainly concerned with data-display functions but attention is increasingly being paid to the provision of interactive facilities.

The Viewdata concept as we now know it began in the Post Office Research Department in 1970/71. The evolution of the original ideas into a working, marketable product has been a team effort, involving the Post Office, terminal suppliers and information providers, but special note must be made of the work of Sam Fedida who is generally acknowledged as the 'father' of Viewdata. The results of this heavy investment over a period of ten years has been PRESTEL, a public Viewdata system now available over two thirds of the country. Although acknowledged to be behind its original commercial targets, PRESTEL can certainly claim to be a technical success; in mid July 1981, PRESTEL supported 10 212 registered sets which make an average of 185 500 data retrievals per day.

Why is Viewdata more cost effective than conventional TP technology? Its chief advantages are:-

(*a*) It reduces terminal costs by use of cheap mass-produced terminals, built around television sets. All sets are built to a common PRESTEL standard

*Note: PRESTEL is the registered Trade Mark of the Post Office (British Telecom) Viewdata Service

thus allowing any terminal to connect to any Viewdata system. Further, the terminals are multi-purpose; they can be used as television or Teletex receivers, as play-back equipment for Video Cassette Recorders or, with additional electronics, as a home computer.

(*b*) The Viewdata terminals available allow colour and simple block graphics to be introduced to the commercial terminal market at an acceptable price.

(*c*) Viewdata attacks network costs and installation delays by

(i) using the existing Public Switched Network rather than dedicated lines

(ii) removing the need for and cost of conventional modems by the use of built in modems within each receiver. Connection to the telephone network is by simple jack plug

(iii) the mainframe resources needed to handle any given number of terminals are minimised by the use of autodial and disconnect

(iv) the human interface to the system is explicitly designed for use by untrained staff.

This last item needs to be explored further. The designers of PRESTEL have clearly done a great deal of work on simplifying the user interface. Very little background knowledge is necessary for a user to use the system; on certain terminals for example it is possible to dial-up, input a username and receive the first screen by a single key depression. Once on the system, all screens should offer guidance on what to do next whilst the syntax of the user inputs themselves is very simple. Interestingly, while many system designers are trying to move towards an 'English' style of input, PRESTEL has gone to the other extreme. All commands are built from the 12 character set 0 to 9,* and #. The system is usable because so few commands are necessary; 7 commands (*0#, *number #, *#, *00, *09, number and #) are adequate for most tasks, so users soon forget about the strangeness of the commands. The use of so few characters has the advantage that it allows very small (and cheap) keyboards to be used; ultimately the keypad of a press-button telephone (which also uses the same 12 characters) can be used in a combined telephone/terminal system.

The technical success of Viewdata as a means of disseminating information is self-evident to anyone who has seen a demonstration of either PRESTEL or BULLETIN; its commercial viability is dependent upon the usefulness and hence value of the information stored within the system. It soon becomes apparent that the major cost in running a Viewdata system is maintaining the accuracy and timeliness of this information.

On *dedicated* systems such as PRESTEL, running on a stand-alone computer, the cost of data maintenance is high because most material is still updated by hand through an editing keyboard. PRESTEL is cost effective since it provides a national audience for the data. The aim of BULLETIN is to make Viewdata systems viable in a smaller, company-wide environment. This can be achieved if Viewdata users can be provided with easy access to the users' already computerised data and applications. The requirement, therefore, is for the *integration* of Viewdata into the

complete DP environment. How this integration is achieved in practice is explained in Section 3.

2 Design criteria and objectives

It must be assumed that the end-users of a Viewdata system are non-specialists who have no desire to become computer experts. It can be assumed however, at least in the UK, that users will be familiar with the PRESTEL interface and, indeed, it seems important for the development of Viewdata as a whole that our interfaces 'reinforce' the PRESTEL style. Like most of its competitors, therefore, BULLETIN has chosen to adopt a terminal interface which utilises not only the PRESTEL hardware interface but also the PRESTEL terminal end user interface. Where BULLETIN goes beyond PRESTEL in facilities, sympathetic extensions of the PRESTEL interfaces have been attempted.

It is worth making the point that most users regard the excellent response times of PRESTEL as an integral part of the user interface. Since BULLETIN supports its terminals through a front-end processor, some compromise has to be accepted here but the maintenance of response times is seen as a vital part of the usability of the overall product.

Below the terminal user interface, any Viewdata system can be regarded as a fairly typical online system. This implies the need for:

(*a*) resilience; the BULLETIN pagestore must withstand failures on terminals, disc drives and in the BULLETIN software itself
(*b*) recovery; after a failure, the system must be capable of restarting its service as soon as possible without the loss of any confirmed data
(*c*) online updating; when editing, the system must guard against simultaneous updating of the same data
(*d*) minimum system operator involvement; it is likely that BULLETIN will be run as a continuous background system
(*e*) HELP facilities; as a system aimed at non expert users, the system must be capable of helping terminal users when they get stuck.

Finally, there were two commercial constraints. Although detailed design did not start until July 1980, there was a requirement for a demonstrable product on the recently announced ME29[1] in January 1981. This implied that implementation must use reliable and well proven design concepts. Further it was recognised that no changes could be made in ME29 firmware over the time period of the first phase of the development. The system was therefore built as a superstructure product with the possibility of using special hardware external to the ME29.

3 The system

3.1 Definitions

Since BULLETIN is required to offer PRESTEL compatibility at the user

terminal level, it is necessary at this point to define the basic Viewdata terminology Since its prime concern is information retrieval, a Viewdata system is centred around a file of screen images, which in BULLETIN is known as the *pagestore.* Each screenful of information is known as a *frame;* while up to 26 frames on the same topic are linked together to form a *page.* Data access is by *page number,* a numeric string, of up to 11 digits, assigned to every page. Requesting a particular page number causes the first frame, known as frame A, of the page to be displayed on the screen. The other frames within the page, labelled B to Z, respectively, can be viewed in sequence by requesting the next frame with the # command. Direct access to non-A frames is not permitted.

Although pages are free-standing entities, pages are related to each other through a numeric tree structure in which the 10 *filials* of a page, formed by adding another digit (0-9) to the right hand end of its page number, are defined as being 'owned' by that page. In BULLETIN, the page ownership system is taken further; a particular page may be defined as a start page or *start node* for an *Information Owner;* all filials of the start page and so on down the tree structure are then defined as being owned by that information owner. These concepts are illustrated by Fig. 1.

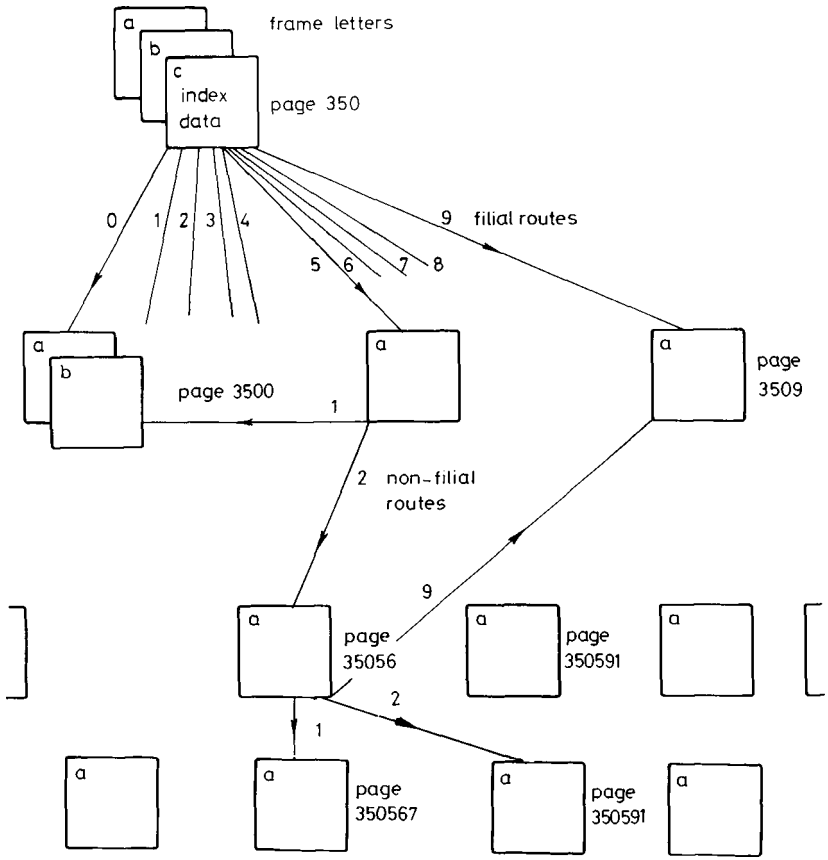


Fig. 1 Viewdata pagestore structures

The terminal user wishing to retrieve data from the system navigates his way through the pagestore with a very simple set of user commands. If the page number required is already known, it may be called up directly. In most cases, however, the user will need to search for the required data; the low numbered pages in a Viewdata system are therefore primarily used for *indexing pages.* As an aid to indexing, a built in *routing* facility is provided on each frame. When a frame is set up, the editor may define up to 10 routes to other page numbers. When the frame is on view, a response of a single digit will cause the system to automatically retrieve frame A of the page referred to in the corresponding entry in the routing table. A response of 0 retrieves the first page referred to in the routing table, a response of 9 retrieves the page pointed to by the last entry, etc. Thus movement through the indexing pages to the data pages is by single-character user responses.

The default routing system follows the filial ownership system; that is the routes on a frame point to frame A of the 10 filials of the current page. It is also possible to define the entries in a routing table explicitly and this allows an indexing frame to steer a user to any part of the pagestore.

In addition to the frame retrieval commands, it is also possible to recall the frame last viewed (*#) and to request the redisplay of the current screen, (xoo).

3.2 The pagestore

3.2.1 The file: The basis of any Viewdata system similar to PRESTEL is its pagestore so the major design decision in implementing BULLETIN was to choose a file structure for the pagestore which would be both reliable and familiar but which would also meet the accessing requirements.

As was described in Section 3.1, access to the pagestore is always by explicit, usually random, page number. However the nature of the filial pagestore system means that the page numbers key set itself will not be random; instead it will comprise dense bands of page numbers separated by large gaps. Thus the most obvious file structure, a hashed random file, was ruled out because of the difficulties of producing an algorithm which would spread the user-defined page numbers evenly across the file. This led to the adoption of a standard index-sequential file maintained to 1900/ME29 Direct Access Housekeeping standards (DAHK) as the basis of the pagestore. This also had the advantage of allowing existing ME29 utilities to be used in the pagestore create/dump function. It was also recognised, however, that conventional index-sequential files have performance problems when used for high-throughput retrieval functions. The target for BULLETIN is to perform no more than a single disc transfer for each frame read, but index-sequential structures require up to 3 index reads plus extra transfers for overflow records. The index problems have been met by the provision of a buffer stack within the BULLETIN Retriever (see Section 3.4). This is maintained on a usage basis so that frequently used index records (and data frames) are retained in store indefinitely. Degradation of the system due to overflow can be prevented by regular use of the BULLETIN dump/reorganise utility. The initial versions of BULLETIN do not meet the target of one transfer per frame but further improvements in the

code are expected to keep the system from becoming disc bound on normal ME29 configurations.

Since index-sequential structures have always been recognised as a less than ideal choice, care has been taken to ensure that the whole pagestore accessing system is easily replaceable. Now that the initial product is completed, it is hoped to spend some time and effort reviewing the possible replacement file structures. This is likely to be either a non-DAHK variant of conventional random files or a straight conversion to the standard database system, known as TME-RAPID, which is now integrated with the operating system.

3.2.2 Record formats: The terminal end-user is aware only of the existence of pages and frames within the pagestore but much 'red-tape' material describing usernames, passwords, security restrictions etc. also has to be stored. All of this information is represented by different record types existing within the single physical file. The convenience to the user of maintaining a single, easily portable, pagestore is seen as being well worth the software complexity necessary to cope with a single file containing 20 different record types.

3.2.3 Screen contents: The frame records within the pagestores contain the colour and graphic control data ready for display on the users terminals. To the outsider, the use of colours and graphics is the most obviously novel feature of Viewdata; in practice it is one of the easiest elements of the systems to handle.

The designers tackled the problem of describing colours and graphics within the ISO character set by redefining the use of the ESC (ISO 1/11) character. ESC is now used to define the start of a control function character pair; the second character defines the function required. Full details are given in the British Telecom specification[2] but as an example ESC followed by A causes all data up to the next control character to be output in red. Other ESC/character sequences allow use of six colours (plus white) for both alpha and graphical displays. All control character pairs occupy a single, usually space-filled, character position on the screen.

When in the graphics mode, the terminal interprets the ISO data characters 2/0 to 3/15 and 6/0 to 7/15 as indexes to a table of 64 graphic shapes. The character set used is compatible with that used by the British TELETEXT services CEEFAX and ORACLE, thus simplifying the building of television sets capable of receiving both Viewdata and Teletext.

The generation of the colours and graphics therefore is a feature of the terminal itself and requires no mainframe intervention. Thus, support of a fully interactive system, in which data is both entered and retrieved through terminals, presents few problems in character handling in that the colour and graphic characters are seen by the mainframe as standard ISO data. The problem areas lie in providing facilities to map existing ME29 files (which use a 64 character set) on to the colour and graphics of a Viewdata frame. The problems in trying to print the Viewdata character set (160 characters and graphics in any of seven colours) on a standard ICL line printer can also be imagined!

It is worth mentioning in passing that the introduction of colour terminals adds a surprising degree of complexity to the design of screen layouts. A good colour display can add life and impact to the most boring information but poor design can confuse and tire users far more than the equivalent monochrome display.

3.3 Handling of Viewdata terminals

The PRESTEL receiver protocol is asynchronous full duplex and cannot be handled directly by any of the existing ME29 couplers. Fortunately, a solution to this problem had already been found as part of the THORNTEL 'special' developed by ICL's Letchworth Development Centre (LDC) for Thorn Television Rentals; the chosen solution was to produce a Viewdata Protocol Convertor based on the LDC CREAM product. The convertor for BULLETIN, known as the BULLETIN Link Unit, (abbreviated to BLU), was originally built around an INTEL 8085 linked to standard ICL boards. This supported up to four Viewdata terminals and converted the PRESTEL format data to ICL C01 protocol for mainframe processing. The Mark 2 BLU, built around an INTEL 8086 and supporting up to 8 Viewdata lines, is now superseding the original BLU.

The BLU performs three functions. First, it controls the content of the terminal screen. The terminal operator has the illusion that data typed in through the keyboard is copied directly on to the screen. In practice, the BLU inspects every character as it is typed and echoes it back to the screen if appropriate. Advantage has been taken of this system to produce an 'intelligent' terminal user interface particularly in the area of screen-editing functions.

Secondly, the BLU acts as a data concentrator. Up to four (eight in the Mark 2 BLU) Viewdata lines are handled and their messages queued for onwards transmission to the ME29 over a 4800 baud line (9600 baud in the Mark 2 BLU). To the ME29, the BLU is seen as a Queuing Line Sharing Adaptor (QLSA) with each Viewdata line addressable as an individual 7181 video terminal.

Finally, the BLU in conjunction with its Model 20 British Telecom modems handles the auto-answer and auto-disconnection of the Viewdata terminals. In sizing terms, an 8 port BLU can be equated with a 7502. Our measurements indicate that four 8-port BLUs can be supported by an ME29/35 with response times of about 2 seconds assuming 9600 baud links to the BLU. The 32 ports in such a system can serve 32 concurrent users so that in a commercial environment some 100 or more Viewdata terminals may use the system on a dial-up basis. Clearly, the maximum number of terminals that can use the system effectively is dependent upon the application and the pattern of calls during peak periods. The ability of a comparatively few BLU ports to handle a large terminal population is one of the main economic strengths of Viewdata vis-a-vis standard TP terminals.

It is also possible to situate the BLU remotely from the ME29, possibly linked to the mainframes by a leased line. This should save telephone costs by allowing a clustered population of terminals to access their BLU at local rather than STD telephone rates.

3.4 *The software*

3.4.1 Basic retrieval functions: Section 3.1 defined the terminology of Viewdata and outlined the range of commands available for data retrieval. These are handled in BULLETIN by the *Retriever,* a free standing TP program which analyses the users' requests and satisfies them with frames of data. The retriever also controls access to the system generally. Access to the retriever is under the scope of a *username;* this may have a password and terminal list defined to limit its use to authorised staff. Once connected to the system, a user may only retrieve pages which explicitly include his username in their access control list; thus a pagestore may contain pages which are freely available, marked as viewable by a username called PUBLIC; pages which are available only to a list of specific usernames; or pages which are available only to the person who created them.

The other main task of the Retriever is the management of the Viewdata terminal network. Since Viewdata terminals compete for ports on the BLU, close control on the state of the system must be maintained if the network is not to become choked by failed lines and abandoned terminals. The Retriever is kept aware of any terminal mishaps by the use of status messages transmitted from each BLU at one minute intervals; in particular they inform the Retriever of any unused terminals or, more accurately, any keyboards which have not been touched in the last minute. If a terminal keyboard is idle for five successive minutes, the Retriever disconnects the port to allow other users access to the system.

Although timeout checks are common features in online systems, the ability of BULLETIN to distinguish between genuinely idle terminals (no keyboard activity) and those being used by slow typists does seem to be a comparatively novel feature.

3.4.2 Retrieval extensions: Although the original concept of Viewdata was that of a data-retrieval system (hence the word Viewdata) it has become increasingly apparent that some degree of interaction is vital to the acceptance of Viewdata for business use. PRESTEL recognised this with the introduction of *response frames;* basically, formatted screens supporting protected and unprotected areas. Because of the weaknesses of the PRESTEL protocol (character level parity but no block level protection against missing characters), great care has to be taken in handling user inputs. PRESTEL and BULLETIN handle this by 'echoing back' the users' input with a request for him to check and confirm that the data received is correct before the system proceeds further. Since the BULLETIN retriever is a free-standing program, all data is written to a journal for later off-line processing by user-written software.

Frame retrieval and response frames exemplify a stand-alone Viewdata system, largely isolated from the rest of the users computer systems. BULLETIN bridges this gap with its 'Window' concept; this provides Viewdata terminal users with access to other user-supplied TP programs. The system works by the user designating one or more frames in the pagestore as *window frames;* such frames are labelled with the name of a user application program. When the terminal user calls up a window frame, the BULLETIN retriever notes the program name and establishes if the named application is currently available. On the ME29, under

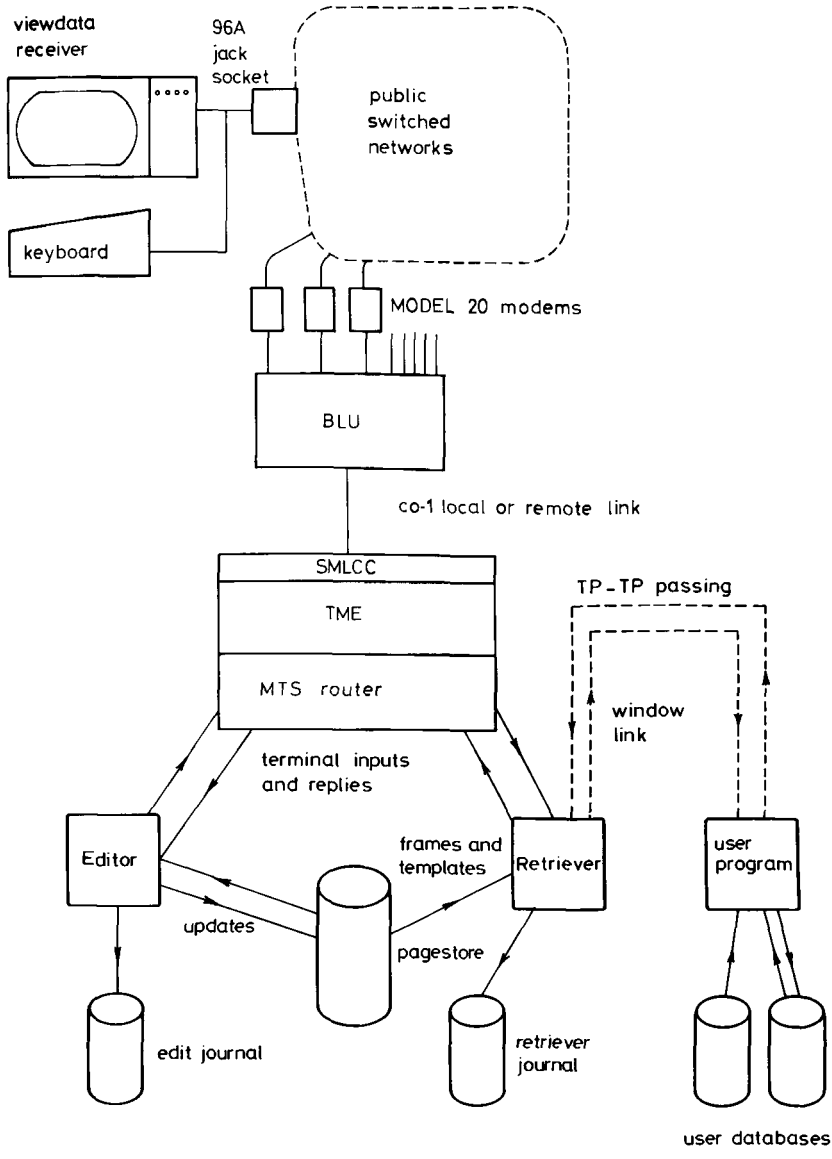


Fig. 2 Schematic of software

MTS, this is implemented by the TP-TP message passing system. If the user application is loaded, the Retriever returns the text of the window frame to the terminal screen; this will inform the user about the application he has called up. All further user inputs are now passed directly to the user application which is responsible for generating a reply to each input message (see Fig. 2). To simplify building a reply, the user application is encouraged to use a frame on the pagestore as a 'template'; application-supplied strings of data can be merged into this before the

complete screen is output. The link between terminal and application can be ended by either end of the link whilst the Retriever applies timeouts to the application to prevent Viewdata terminals becoming locked out due to looping or failed applications.

The uses of this system are legion. Specific new programs could be produced to provide, say, keyword search on one of more files, or to perform complex calculations on data supplied by the terminal user. More importantly, existing TP systems may also be modified to support enquiries from Viewdata terminals. In an order entry system, for example, window links could be added to provide enquiry facilities for home based salesmen to query the state of their orders, whilst it would also be possible to provide a management facility for enquiries on orders received so far today. It is hoped to adapt many of the existing ICL packages to provide such enquiry systems. Carried to its logical conclusion, this system converts Viewdata into a TP monitor in which the system handling aspects (log-on, terminal fails, recovery, even charging by use) are performed by BULLETIN whilst the application code is written by the user.

3.4.3 Updating the pagestore: It seems possible that some BULLETIN systems will consist solely of a Retriever linked to one or more window programs. In this situation the pagestore will be used purely to hold template pages. In most installations, however, some if not all of the information will be held as standard text frames in the pagestore. BULLETIN offers three techniques for maintaining the pagestore.

3.4.3.1 The Editor: The pagestore can be updated by an interactive editor that runs alongside the Retriever. The editor operates directly on the pagestore and includes the usual elements expected of any online system; it uses a locking system to prevent simultaneous updating of a page, while intermediate results are kept on a workfile until the terminal user confirms that the edit is complete. This is necessary since the Editor has no control over the actions of the retrieval software; the pagestore must be updated in such a way that the Retriever always gets coherent results even though the page it is retrieving is currently being edited. All edits are logged into a journal. This can be used to rebuild the complete pagestore from a previous dump if the pagestore becomes physically corrupt.

Designing the user interface to the Editor required major extensions to the PRESTEL interface. Again it had to be assumed that non-computer staff would use the editor, so a menu-select interface was adopted to guide users through the system to the appropriate point. In practice this turned out to be inefficient since it has proved impractical to standardise the menu screens to any significant degree. This hinders the experienced user who is forced into reading every screen. Further versions of the editor will therefore move to a 'keyword' style of operation, at least at the lowest levels of the Editor. About 10 keywords or combinations of keywords should be sufficient to allow the experienced user to move from any editing task to any other with a single command. Non-expert users are catered for by a structured set of HELP screens.

When editing frames the Editor operates in screen mode. To allow colours and graphics to be defined, a larger 'editing keyboard' is necessary. The extended

character set of Viewdata adds considerably to the time taken to input frames; an average rate of five frames per hour can be assumed on graphic data. This has led to pressure for the development of intelligent editing systems, of which quite a number are now available. They all consist of a micro with floppy discs driving one or more screens and keyboards. BULLETIN is currently unable to support these devices, so similar, though less sophisticated facilities have been provided within the BLU. The following features are currently available:

- line insert and delete
- character insert and delete
- change all control characters (eg change all blue text to yellow text)
- move a rectangle of data from one part of the screen to another
- store an area of the screen as a macro which can be recalled later on
- move graphics half a character

3.4.3.2 Bulk formatting: The costs and complexity of inputting data by hand through a keyboard were described in the previous Section. Where data already exists in the ME29 it is obviously preferable to transfer it directly to the pagestore. This task is carried out by a suite of programs known as the Bulk Formatter.

Bulk formatting, as the title suggests, produces a number of pages to a similar layout with just the individual details on each page being different. Typically, a template page is used and then the details of each record are placed on it to produce the formatted pages.

Bulk formatting can potentially accept any ME29 disc file of serial or index sequential organisation, identify record types within that file and extract all the records of a given type. In addition, individual parts of records (fields) can be extracted so that the whole record does not need to be formatted. These fields can be character, numeric, or binary integers.

Each of these fields can then have all the display attributes of Viewdata associated with them e.g. colour, flashing, background colour, double height. The position on the Viewdata screen at which these fields are to appear must also be defined.

The pages created by this method can also have information displayed which did not come from the record, but this is static information in as much as that it will be identical on every frame created.

All the information required to specify the file extraction and page formatting is supplied as a parameter set that is input and amended by a TP program driven by a standard video.

Having converted the information itself to a BULLETIN format, a Viewdata routing system to make the information easily accessible is created. This is achieved by producing index pages with explicitly defined routing to steer the user to any of the formatted pages. The user specifies the field in the record on which the index is to be built.

An example of the use of bulk formatting is to transfer personnel information from a personnel file sequenced on staff number on to the pagestore, with an index being constructed on the surname of each staff member. An extension to this would be to index the pages on surname within department.

Further facilities are available to make the displaying on the screen more flexible. If, for example, the text goes beyond a line, then the text will be wrapped around to the next line with the split occurring on a word boundary. Margins can also be specified to control where the field is displayed. Subsequent fields can then be displayed in positions relative to the end of the previous field rather than as an absolute position on the screen.

As the structures of many files were designed with one particular application in mind, not all files will be in a form that can be bulk formatted immediately. These files would require some pre-processing and although this is extra work, the Bulk Formatter will still do the major job of embedding all the Viewdata attributes and building the routing system.

Once a parameter set has been defined to format a file then the file may well have more up-to-date information placed on it by another computer application. Assuming that the parameter set is still adequate, all that is required is for a batch system of programs to be rerun to update the pagestore to reflect the file amendments. This could be done as an overnight exercise.

3.4.4.3 Pagestore updating by program: Finally, BULLETIN provides two ways of directly updating the pagestore by user written program. The first is a subroutine suitable for inclusion in a COBOL program which allows a user-written program to read or write frames. This routine cannot be used concurrently with the rest of the BULLETIN software, so is mainly of use for off-line batch programs. An example of use would be in a BULLETIN accounting suite, run monthly, where the last program in the suite could copy a summary of the results on to a well-known page on the pagestore for general viewing.

The second technique allows TP programs to update the pagestore concurrently with live use of the system. This is achieved by the user program passing messages to and from the BULLETIN editor which performs the reads and updates on its behalf.

4 The future

The system described in this article represents a basic Viewdata system: a pagestore which can be viewed plus the tools for updating it. Viewdata technology is developing rapidly and continued development of BULLETIN is likely to include:

Non-ME29 environments
Since the MTS communication system is supported by DME/2 systems it has been possible to make BULLETIN available on 2946. Although no software

development is necessary to effect the conversion of the basic package, the long term implications of a general move to larger machines with their higher performance expectations have yet to be fully analysed. Similarly, no detailed assessment has yet been made of running comparable Viewdata software under other ICL operating systems.

Gateways

Gateways are currently the major development area in Viewdata. The concept is simple enough; the national Viewdata service (PRESTEL in the UK) acts as the centre of a star network of computers. All users connect to PRESTEL and call up frames as normal. Certain frames are marked as *Gateway frames.* When these are retrieved, PRESTEL establishes contact with a remotely situated *external computer* owned by an outside organisation. All further user inputs are now passed to the external computer which must respond to each message with a frame similar to that of PRESTEL. The link between PRESTEL and the external computer is via the X25 packet switching service of the country concerned.

So far three countries, Britain, Holland and The Federal Republic of Germany have announced gateway systems, all of which are basically compatible. Although pilot schemes planned (the West German system has actually been running for a year) are all relatively small (about eight machines each), the potential market across Europe as a whole runs into several hundred systems. It is therefore likely that the BULLETIN retriever will be enhanced so that it can handle the gateway message protocol. This will allow any PRESTEL user to gain direct access to the frames of data on the BULLETIN pagestore.

The potential of this system is enormous. To the owner of an external computer, buying a gateway link into PRESTEL gains all PRESTEL users (currently about 10 000) as potential viewers of his data. Equally, since all PRESTEL links are charged at local charge rates, any widely distributed organisation gains a very cheap national online system. This aspect is made more attractive by the likelihood of links emerging between the various European Viewdata systems, in particular the British, Dutch and West German systems.

Message Systems

A natural extension to displaying frames of text is to provide a mechanism for sending messages between BULLETIN terminal users. This is expecially necessary since many users will be widely dispersed and may rely on Viewdata as their main means of contact with their headquarters: salesmen based at home for example. However it is recognised that message systems are not the exclusive preserve of Viewdata systems and that links with other products must be provided. In particular it seems sensible to provide for the interchange of messages and indeed documents between BULLETIN and WORDSKIL. Such linked systems fore-shadow the development of more generalised office automation products which may replace existing Viewdata and word processing systems by a single integrated product.

Conclusion

The development of BULLETIN is an important step in the path of simplifica-

tion of access to computer-held information. The adoption of standards based on those of PRESTEL gives BULLETIN users inexpensive terminal equipment and simple procedures which can be combined to offer effective on-line information services to the day-to-day operations of an enterprise.

References

1 LAKIN, R.: "ME29 Initial Program Load – an exercise in defensive programming" *ICL Technical Journal,* May 1980, **2(1),** pp.29-48.

2 Prestel Bulk Update, Technical Specification, Appendix F.

Development philosophy and fundamental processing concepts of the ICL Rapid Application Development System RADS

A.P.G. Brown, H.G. Cosh and D.J.L. Gradwell
ICL Distributed Systems Development Division, Bracknell, Berkshire, UK

Abstract

This paper describes the Rapid Application Development System (RADS) which is currently being specified and studied within ICL. It includes a new programming language which is considerably more problem-oriented than languages such as COBOL and which is intended to give significant gains in productivity.

Unlike a number of non-procedural languages, RADS includes a complete computational model, allowing its use, ultimately, for any programming problem. A major part of this paper is devoted to describing this processing model and to contrasting the facilities it provides with the acknowledged problems of conventional programming languages. The new model presents the compiler designer with new problems, some of which have been solved in a prototype system.

The target system is intended to combine the facilities provided in the basic RADS model with higher-level functions, and with the facilities of the Data Dictionary upon which RADS is based. This it is hoped will reduce by an order of magnitude the costs of developing and maintaining an application. Experience with a subset of the language for report writing confirms our belief that this is achievable.

1 Introduction

ICL's Rapid Application Development System (RADS) is a new programming language, integrated with a Data Dictionary System, designed to support the development of commercial data processing applications. These may be transaction processing or batch oriented, with data held in a Codasyl database or in COBOL files. Together with ICL:s Data Dictionary System (DDS)[1] RADS will support the specification, documentation, coding, compilation and maintenance of the

user's application systems. The prime objective of RADS is to improve the productivity of systems designers and programmers.

This paper explains the growing software crisis and the need for an order of magnitude improvement in productivity over present day practices. It explains the design goals of RADS and the way RADS is intended to fit together with other ICL software products.

The major part of the paper then discusses the concepts behind the RADS language, particularly the method of describing processes.

There are several alternative or possibly complementary approaches to improving productivity in application development. Examples are specialist application languages, applications packages, macro systems and framework programs. All of these will probably be used with varying levels of user acceptance. The RADS approach differs because it involves a new style of programming language and a departure from the von Neumann machine which underlies conventional languages[2].

The reader should note that this paper does not describe a user interface, but rather, discusses the fundamentals of a new processing model.

We believe that the RADS processing model will provide the basis for the implementation of a more intelligent compilation system, able to yield the desired level of productivity gains, while supporting the development of general-purpose programs.

2 Why develop a new programming language?

2.1 The software crisis

The last two decades have seen an astonishing rate of change in the cost performance of computer hardware both in raw computation power and in information storage. Technology improvements have halved the cost of hardware approximately every two and a half years and there is every indication that this rate of change will continue or even increase.

The consequence of this technology change is that the users will have much more computer power available and that quite small enterprises will be able to afford machines which a short while ago would be considered to be large computers.

The decreases in hardware cost is not being met by an equivalent decrease in man-costs, so that the cost of systems development and programming is rapidly becoming the predominant element of users investment. By the end of this decade the hardware costs could be less than 10% of users' investment in computer systems.

An analysis of users' data-processing departments today shows that, even after many years experience, our ability to develop new systems quickly and at low risk leaves much to be desired[3]. System development costs are not easy to estimate.

Many projects run late or fail to meet their functional objectives, and many installations spend most of their development resources on the maintenance of old systems.

While there is no doubt that our knowledge has increased and that computer scientists now have a much wider and deeper training there is still a critical shortage of skilled staff. Try as we may, there is no way that we can increase the supply of staff, especially experienced staff, to keep up with the supply of computing equipment.

One solution is for users to purchase their systems ready made, and to adapt their operations to the constraints of an applications package. No doubt package suppliers will enjoy an increase in business, but the fundamental problem of the cost of software development remains.

We must look at ways of providing systems development staff with better tools so that they can improve their own productivity. It is clearly cost-effective to use more computer power to achieve this end.

Two of the factors contributing significantly to a programmer's productivity are his programming language and the associated development environment. In the recent ADA studies[4,5] it is significant that, for the first time, as much effort is being expended on the development environment as on the language itself. We too, believe that the development environment is of strategic importance, and have therefore designed RADS to work with a data dictionary database, containing all the design information about the data and the applications. However, the Data Dictionary System has been well defined elsewhere[1] and this paper concentrates on the programming language itself.

2.2 Developments in procedural programming languages

At present the most widely used programming language is COBOL, the first version of which was developed in 1959. Since then there have been many extensions and improvements but the net effect on productivity is small compared to the improvements from hardware developments. Of course compilers give better service because the hardware is faster and they can use more store, but the innate effectiveness of the computer languages has not changed. Hence, it is nearly as difficult to write a COBOL program now as it was in 1960.

The languages PL1 and Ada represent attempts to bring programming up to date by incorporating the good features of other languages into a new language, and thus escaping the forwards-compatibility constraints of existing languages. Both attempts have resulted in very large languages requiring expensive compilers but the productivity improvement compared with their predecessors (COBOL, Fortran, Algol, Pascal), is probably of the order of a few percentage points.

Systems like FIND 2, MARK IV and RPG2 achieve higher productivity, but at the cost of loss of generality. Any application can be tackled as long as it fits the

constraints of the run-time program framework. There are many such application-utility systems, each with its own merits and restrictions. They all do some jobs well but all lack a general computation facility that would enable them to challenge the standard programming languages.

2.3 Structured programming

Some users have achieved increases in productivity through structured programming[6] which imposes a discipline on the programmer and improves his ability to write error-free code. This is interesting because it appears to demonstrate that the richness and variety of computer languages is both unnecessary and counter-productive. Structured programming has influenced Ada and the COBOL 82 proposals [5,7] but its use is permissive. Programmers are still free to create unstructured programs if they wish.

2.4 Non-procedural languages

In a conventional procedural language like COBOL a programmer has to be very aware of the sequence in which his program statements will be executed. Each statement makes a small change in the state of the process and the programmer has to ensure that these changes occur in the correct sequence to satisfy the program requirement. It is easy to lose the thread of a program because of the lack of structure and the need for attention to detail.

There have been a number of developments that try to raise the level at which the programmer thinks, and to remove unnecessary procedurality from the programming language. For example, report writers like FIND 2 restrict the programming task to data selection and formatting. A pre-determined program structure is used as a basis for the actual program, saving the programmer from having to design and code it.

Another interesting example is the LUCID language[8], where the order of statements is irrelevant, assignments are statements of identity, that is, mathematical equations, and a variable may only have one assignment statement in the program. The benefits of this approach are discussed later in Section 4.2.

3 Relationship of RADS to other software

The RADS language is intended primarily for use in the development of commercial data processing systems. It is designed for the implementation of transaction processing and batch applications which may access data held in a Codasyl (IDMS) database or in conventional files.

Because of the prevalence of COBOL systems the file standards are deliberately compatible with COBOL, and all COBOL data item formats are supported.

The language includes facilities for defining the structure and format of files, databases, interactive TP screens and printed reports, as well as for defining the logic of the processing to be done by the application.

RADS is not an interactive system in the sense of a query facility, which responds to a query with data values. RADS allows a designer or programmer to work interactively but the output from RADS is an object program which forms a component of his application system.

3.1 The Target environment for application development

Fig. 1 is a schematic diagram of the RADS system showing its relationship to the data dictionary database. The application definition is built up by the dictionary user interacting with the dictionary software, which updates the dictionary database for him. He has a range of tools available to him to invoke various generators, including the RADS program compiler. All of these tools share common definitions of the database and files. The tools generate object code that form the user's application run-time system.

Basing the RADS system on the Data Dictionary System provides the user with facilities to document his systems analysis (his data model and the events and functions in the company), his database and file design and any programs written in COBOL.

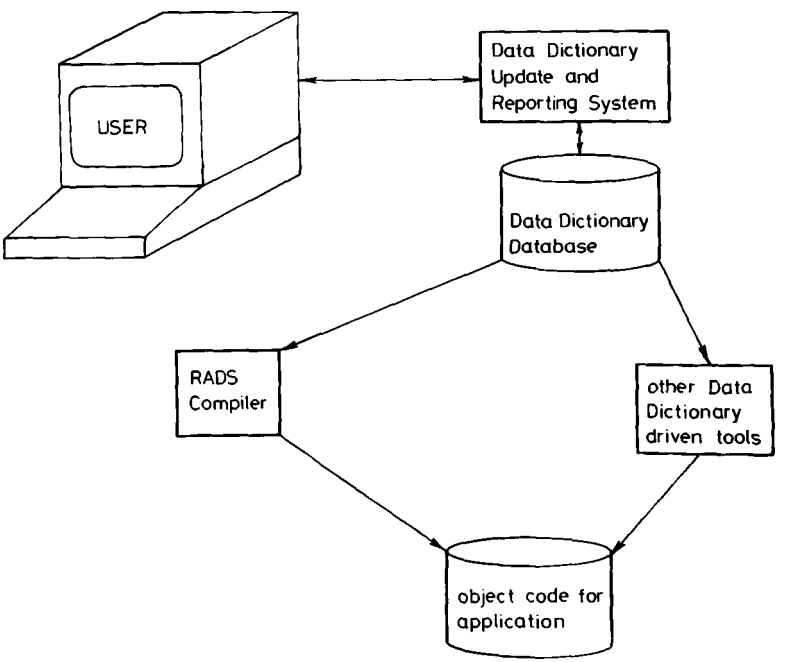


Fig. 1 RADS architecture

The dictionary provides a number of facilities important to a program development environment:

(*a*) A system or program definition can be constructed incrementally and does not have to be written in a prescribed sequence

(*b*) The DDS is interactive. It can respond to simple errors and prompt for omissions while the user is at the terminal

(*c*) The DDS allows the user to query the program, getting different displays of its structure or of the interactions between its components

(*d*) The DDS contains data descriptions that are used by COBOL and IDMS, as well as RADS. These form a library of code that is available for use directly

(*e*) The DDS facilities for authorisation of access to definitions and commands and for version control assist with the development and controlled maintenance of systems

(*f*) The documentation of reporting facilities within DDS make separate system and program documentation unnecessary.

4 The RADS language

This section provides a brief overview of RADS facilities and introduces the language design philosophy. There are frequent comparisons with COBOL because RADS is expected to be used most often in a COBOL shop as either complementary to or instead of the COBOL language.

The RADS language represents either a large jump or possibly a step in a different direction from that in which the conventional languages are seen to be developing. It combines the benefits of a non-procedural programming language with the disciplines of structured programming and clear data structure definition. It provides sufficient procedurality to model the program's interactions with the real world. At the same time, many of the programmer's normal tasks are taken over by the compiler allowing him to concentrate on the logic of what he is trying to achieve.

We believe that the long-term solution to the productivity problem is to raise the level at which the programmer thinks, and to make compilers that are more intelligent and are better able to map the program definition to the underlying hardware.

4.1 Language facility design

At the heart of RADS there is a processing model which provides the foundation for the semantics of all RADS facilities. A RADS process is defined as a Jackson Structure[6] and can be envisaged either as a sequential process or as a data object.

This duality serves to clarify the semantics and provide a graphic picture of the execution of a process.

The basic model forces the programmer to structure his problem and to concentrate on structure before details. There are no control-transfer instructions such as GOTOs or BRANCHES in RADS because this information is represented statically in the structure.

A second basic difference is that all arithmetic or conditional expressions in RADS contain references to values which are at related nodes in the data structure. In procedural languages such references refer (indirectly) to positions in memory which hopefully contain the value the programmer wishes to use. In RADS the allocation of memory space to support required values is entirely the responsibility of the compiler and this common source of programming errors is removed.

In the development of a new language a sound basic processing model is essential to guarantee that the language can be used for any application. Systems without such a general model can be expected to run out of steam when the requirements become too complex for them to handle. This is the case with certain utility systems, framework programs and the like.

The RADS basic processing model can be used to solve quite complex problems but, for the general application programmer, further facilities can be defined which are more attuned to specific problems. In RADS these facilities are defined (by the designers, but not necessarily in user manuals) in terms of the basic underlying model. We can distinguish two levels of higher facility. One is a simple shorthand like a macro facility. The second is a prescriptive statement where the user specifies the result to be achieved and the compiler constructs the program accordingly.

For usability, many details can be left to language defaults, but it is important to maintain a consistent guessable approach to them. This is a complex area where care must be taken to maintain a consistent user view and avoid surprises. It is considered important that we should not add facilities in higher level constructs that cannot be accomplished using lower level facilities, as this would conflict with the requirement for generality.

4.2 Input, output and databases

The COBOL programmer has to execute READ and WRITE statements to control files and printing. In a database environment he has to control a complex interface using many other verbs to navigate within the data structures. He has to ensure that records are read and written at the appropriate times and pay attention to the contents of various currency indicators.

The RADS user has no I/O commands. His definition of the process makes reference to the content of files or databases as if the data were part of his process, which it is. It is the compiler's task to determine when, and for how long, data is required in memory and this can be deduced from the structure of the process.

The definition of data required by a program is declarative. The programmer defines a structure which can be matched or mapped to the database. This is supplemented by selection data so that the structure plus selection determines both the record types and occurrences required.

The integration of persistent data objects (that is, those that survive after the execution of the process is completed) into the RADS process simplifies the definition and eliminates the Data Manipulation Language (DML) which is a common source of errors. It also makes possible the specification of more complex validation in the database description. It is widely agreed that such validation criteria belong with the data description, where they need be specified once only. But current 'record at a time' DMLs would not allow a constraint such as 'this set must have at least two members'. Database processing in RADS is the subject of a separate paper.[11]

4.3 Transaction processing

The COBOL programmer is constantly aware of the functions of his program and of the TP monitor. He has to plan his program to receive a message, process it, store any intermediate results in a slot file and return a message to the terminal. The system designer has to design a transaction by chopping it into a set of interactions and defining the interfaces between them.

In RADS the unit of design is larger. The system designer starts by designing the structure of a complete system – top down. If he wishes, a whole service may be defined as a single structure. A service may be structured into a hierarchy of applications or transactions processed by different programs but this structuring is for administration and tuning purposes and does not affect the logic of the system.

The run-time RADS system handles all the interfaces between the compiled RADS program and the TP monitor, including storage of intermediate results. It is the compiler's task to allocate memory and to analyse its use to determine the data to be saved between interactions.

4.4 Formatting

Printed reports and VDU screens are essentially two-dimensional media. In designing report formats or screen layouts the designer typically takes a layout form and draws boxes to contain particular elements. The mapping of data objects such as records and items onto these formats is a tricky and tedious task for the programmer because of the wealth of boring detail and the fact that lines must be set up for display one by one. This is why report writers enjoy such popularity, even though their facilities are often idiosyncratic and restricted.

RADS has a clear model of formatted data objects which are envisaged as two-dimensional blocks of media. A report may be planned as a structure of component reports each composed of a structure of pages. Each page may be envisaged as a layout of two-dimensional blocks. The contents of the blocks can be mapped or

matched with the data within a file, enabling RADS to generate a program to print the file (or the relevant part of it) in the requested format.

Report formatting and screen formatting facilities are intended to be used along with standard processing facilities. Much of the formatting information can be defaulted, or defined once in the dictionary, so that the task of preparation of reports becomes primarily that of defining the required content.

5 Describing processing in RADS

5.1 Structuring

RADS makes extensive use of Jackson structures[6,10] to define the structure of all complex entities known to the system. In particular files, databases, database views and RADS processes are understood by RADS through their structures. Printed reports and interactive processes are also structured in the same way but their consideration is beyond the scope of this paper.

The Jackson structuring methods were adopted by RADS designers because the decompositions allowed are necessary and sufficient for the definition of all classes of hierarchical structure. Since a hierarchy is the most complex structure that can be mapped onto a sequence and since any program is in some sense (time) sequential all structures processed by a program can be viewed as hierarchies by the program. This topic is considered further in Brown *et al*[11], which discusses sequential views of network and relational databases.

The language constructs for describing data and process structures differ because for data we are concerned with recognising the structure and with process structures we are generating the structure. Thus in a data structure a choice may mean that at this point the viewer can expect to find a debit record or a credit record. In the process structure the conditions for generating each record type would be specified.

Before discussion of process structures we give a brief explanation of Jackson structures. In the abstract we define the structure to consist of nodes, without attaching any meaning to the notion of node. A node is either a terminal node or a composite node. A composite node is composed of other nodes in a defined structure. A terminal node is complete in itself and has no (known) substructure.

It is important to understand that the following discussion on Jackson structures and diagrams relates to *types* of object. When we define the structure of a node called X, say, we are defining a particular characteristic of a set of real data objects that conform to that structure. It is the same distinction that data administrators make between record types (defined by record descriptions), and the set of actual records that conform to the description and contain real values (held in the database).

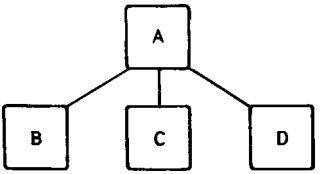


Fig. 2 Single-level decomposition of A

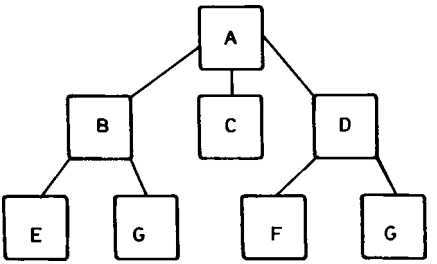


Fig. 3 Two-level decomposition of A

Figs. 2 and 3 show a single level and a two level-decomposition of a node, respectively.
In the diagram the nodes are named. In a complete non-recursive structure the same node may appear at more than one position, but no node may be a component (direct or indirect) of itself.

Note that each node is a direct component of precisely one other node with the exception of the origin or root node.

5.1.1 Types of decomposition: There are precisely four ways in which a node may be decomposed into its components. These four basic decompositions enable any composite structure to be resolved into its components but, compared with other techniques, it may be necessary to introduce extra nodes into the structure.

The four decompositions are:

1. Grouping (of nodes in an arbitrary order)
2. Sequence (of nodes in a specific order)
3. Choice or alternative
4. Iteration (repeated occurrences of the same node)

These decompositions may be represented diagrammatically as in Fig. 4.

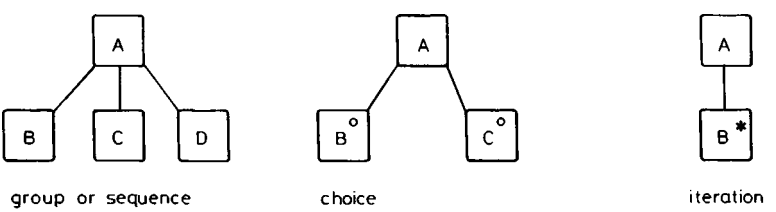


Fig. 4 Diagramming conventions

The meaning of the types of decomposition may depend on the type of object being modelled. For example sequence may represent an ordering in time (B then C then D) or it may have spatial significance (B next to C next to D). Similarly an iteration may represent repeated events or computations or it may mean that there are multiple occurrences of an object.

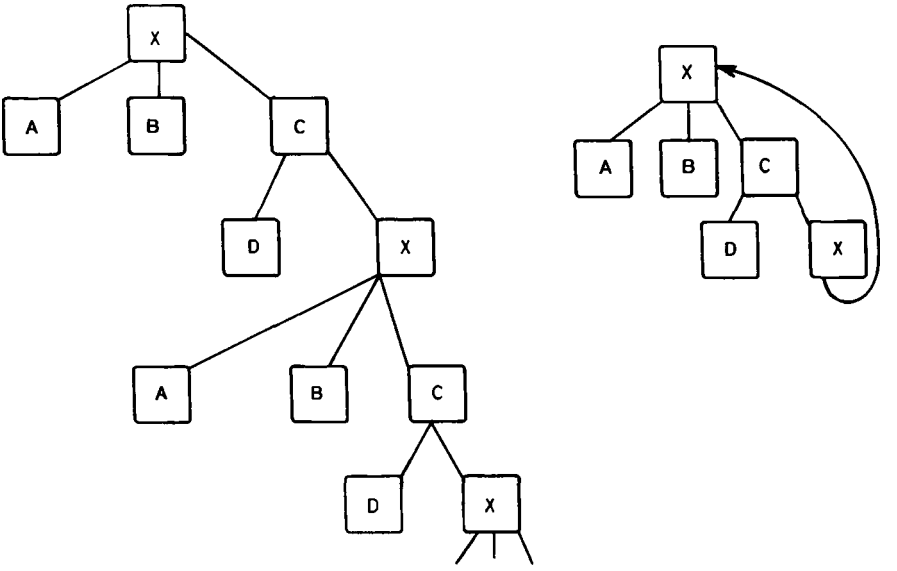


Fig. 5 Recursive structures

5.1.2 Recursive structures: A structure is said to be recursive if the decomposition of the structure includes a node of the same type (with the same structure) as the original node. This is illustrated in Fig. 5.

Recursive structures can only terminate if the recursive component is either an iterate or a component of a choice. This is shown in the two diagrams below (Fig.6). These structures terminate when the C iteration is empty or when the D option is present.

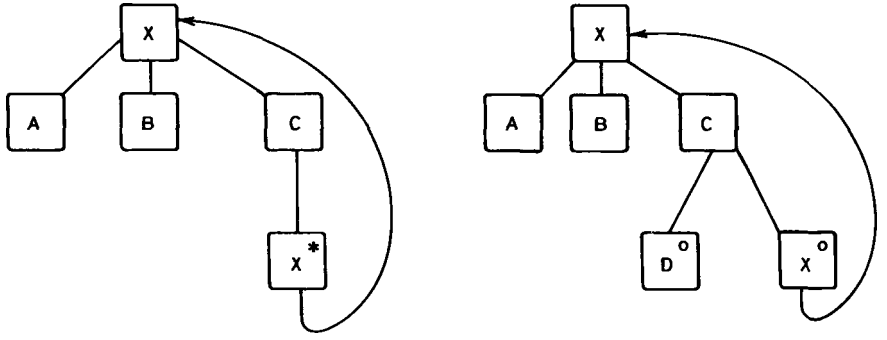


Fig. 6 Termination of recursive structures

5.2.1 The basic structure of a process: In the process model for a conventional language the program is executed in the order in which statements are written except when control is transferred by branch instructions. As was pointed out in the papers on structured programming[6,12,13], it is possible and desirable to construct programs using a restrictive set of control structures.

The structured programming techniques were evolved particularly to address the problems of GO TO statements and data scope. The GO TO problem can be appreciated by observing that branches into the instruction sequence from other parts of a program confuse the structure and the programmer and make that part of the program difficult to check and to maintain.

A RADS process structure is defined only in terms of the data objects that the process wishes to create. It is the compiler's responsibility to determine what procedural code is necessary to construct these data objects, and to determine the order in which it will be obeyed.

Each process has an interface which defines the external view of the process. This interface is a data structure. Components of it are labelled INPUT, OUTPUT or UPDATE depending on whether they are provided externally, created by the process, or provided externally and updated by the process.

Each process may also have a private local structure that is not externally visible. The structure of the complete process is a grouping of its interface structure and any local structure it may have.

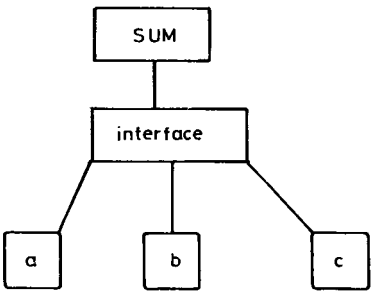
The following example (Fig. 7) shows the text definition (i.e. what the user actually writes, or enters on the keyboard) and the resulting structure of a rather trivial module that only has an interface. It adds two numbers together. In this example

Text definition

```
MODULE sum
*INTERFACE ITEM a INPUT,
           ITEM b INPUT,
           ITEM c OUTPUT

 FOR a,b,c
*TYPE integar
 FOR c
*FROM a+b
```

Diagram of the structure



*FROM a+b

Fig. 7 MODULE sum

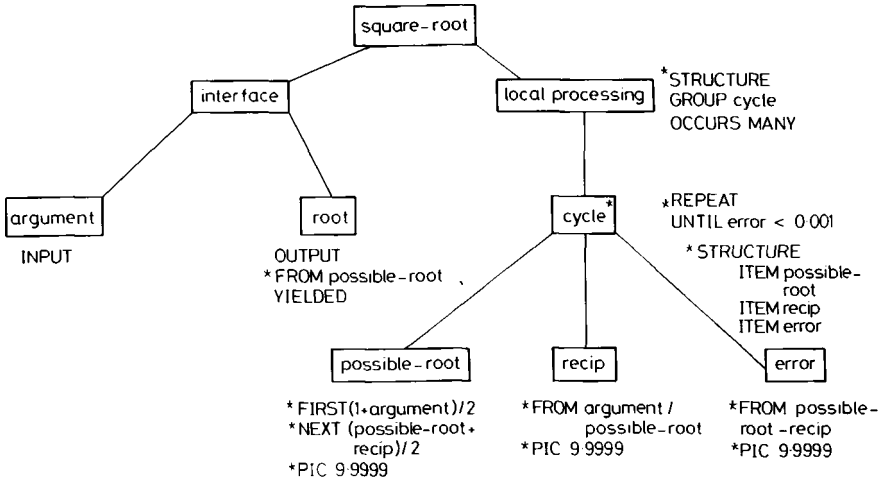


Fig. 8 Square-root module

and those following the Data Dictionary System convention is used, in which a star preceding a word (e.g. *TYPE) indicates a property. The use of the same symbol to indicate repetition, as in Figs. 6 and 8, should not cause confusion.

We shall shortly consider modules with local structures as well.

5.2.2 Assigning values to nodes: We know that the definition of the above module is complete because the source of the value of each node is defined. Nodes that are INPUT are given values externally to the module. Other nodes have a *FROM property which assigns a value.

As in LUCID[8] a node has, basically only a single value. (We shall see when considering updating that a NEW value is needed as well as an OLD value.) This contrasts with a language like COBOL where an item may be given different values in different ways in different parts of the program. We view the possibility of multiple assignment statements for the same variable with the same disdain as Dijkstra [12] viewed GO TO statements. In such a program, it is not clear when the value is initiated, which parts of the program can change it and when the value should be frozen or set back to unassigned. If statements that assign new values to an item are scattered throughout a program it is very difficult for the maintainer of that program to know what the item means at any point in the program.

Most programmers work from flow charts, which can be considered as a model of the control structure of the program. There is no equivalent model of the data, showing when values are generated and when they are changed.

Clearly, if a node has only one value these problems are all avoided. The RADS designer defines the meaning of each data item and its derivation in terms of other items within its scope. The problem of determining when the value is to be calculated

is of no interest since it does not affect the answer and the compiler is given maximum freedom to determine a suitable or an optimum sequence for generating node values. It should be said however that the concept of a variable is helpful if the module is to interact with the outside world during its execution. This will be discussed further in Section 5.3.2.

5.2.3 Iterations in a RADS program: The RADS program structure can contain iterations, which are similar to procedural language loops. Where a procedural model involves changing a given set of values on each repetition until they satisfy some condition the RADS process model generates a number of iterates, with the values of each iterate being available to the rest of the process.

In a generated iteration the last iterate has a special significance and is termed the yielded iterate. As with the model for assignment, the RADS iteration model is similar to that of LUCID.

As an example, we show a module to calculate a square root.

On its interface is the argument whose root is to be found. This is an INPUT item. There is also an OUTPUT item for the resulting root. A local processing structure is needed that consists of an iteration of possible roots, reciprocals and errors that are generated until an iterate occurs which is sufficiently close to the true value.

References within expressions on nodes of the iterate (here called 'cycle'), can refer to nodes at the same level or to nodes higher up the structure. References at the same level refer to the value of the item in the same iterate.

As in LUCID, *FIRST specifies the way in which the value of 'possible-root' is to be computed on the first iterate of 'cycle'. *NEXT specifies the way in which the value of 'possible-root' is to be computed on the successive iterates. In this case the 'next' value of 'possible-root' contains a reference to the current value of 'possible-root'. The 'next' value calculated on one iterate becomes the actual value on the next iterate.

For each iterate the values of root, recip and error are generated. Then if error $<0{\cdot}001$ this iterate becomes the yielded iterate and no *NEXT value is computed, otherwise the next iterate is generated. The yielded iterate is not part of the iteration and cannot be referenced by subscription or indexing into it. In the above example if argument were 1 the generated iteration would be empty.

The order of the boxes in the diagram is not significant. The order in which values can be generated is entirely dependent on the arithmetic expressions although iterates are clearly generated in order.

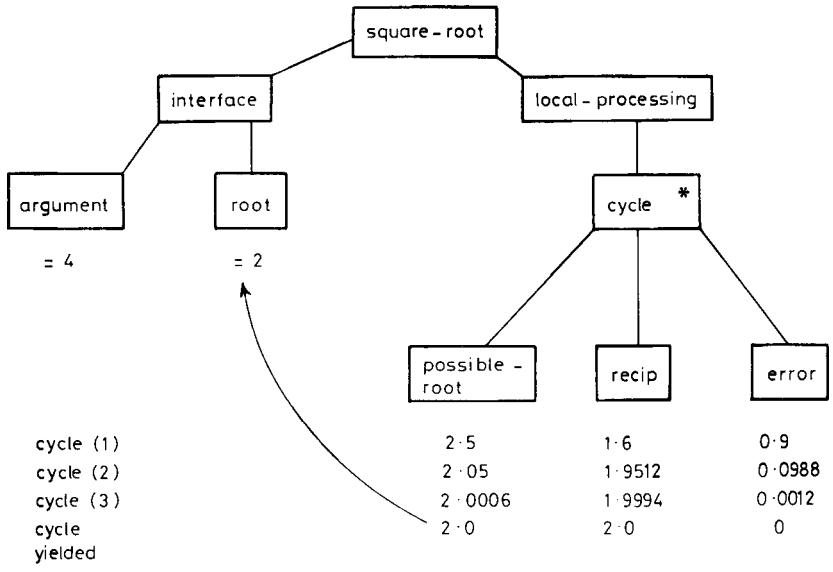


Fig. 9 Square-root module execution

Fig. 9 shows an execution of the process with values given.

5.2.4 Choices in a RADS program: The RADS program structure can contain choices. Fig. 10 shows the 'Daily Batch Control' module that invokes the 'End of month run' module on the first of each month, but otherwise invokes the normal daily run. Module invocation is discussed in more detail in Section 5.2.6.

Thus we have seen that the local processing structure of a module may have any structure built of sequences, iterations and choices to any level of complexity. Likewise, the interface may be of arbitrary complexity possibly consisting of several items, several files and a database.

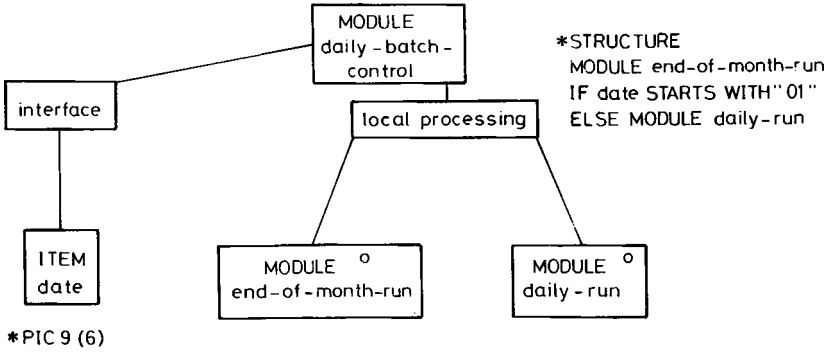


Fig. 10 Daily Batch Control Module

5.2.5 Name references: In the previous examples, *FIRST, *NEXT and *FROM are properties of derived nodes. The syntax of these properties is an expression which, for numeric items, is an arithmetic expression. Such expressions in most languages contain references to named locations in working storage. RADS expressions contain value references, which refer to nodes within the process structure. These nodes have actual values and are not thought of as locations in working storage although the compiler may have to allocate space to the object program.

There are conventions for determining which node is referenced by a given name. For example the references in Fig. 8 in the 'error' node refer to the values within the same iterate. The top node, 'square-root' could be referenced from 'cycle error' but would produce a compilation error diagnostic because the program would have a circular definition.

It is possible to reference the value at any node in a process provided that the node can be referenced unambiguously from the referencing point. There is a notional path through the structure from the reference to the referenced node as in Fig. 11.

In Fig. 11 a reference at X to H would have to indicate which iterate G was to be chosen. A reference to S would be valid if the P option were present. If Q were present the referenced value would not exist. It is more usual for such references to be within choice-expressions so that the value to be used when the option is not present can be stated. For example we might specify

```
*FROM P. S  IF P ELSE 24 IF Q
```

The *FROM property need not specify an arithmetic expression. It is possible to specify any expression from any node so long as the result of the expression is

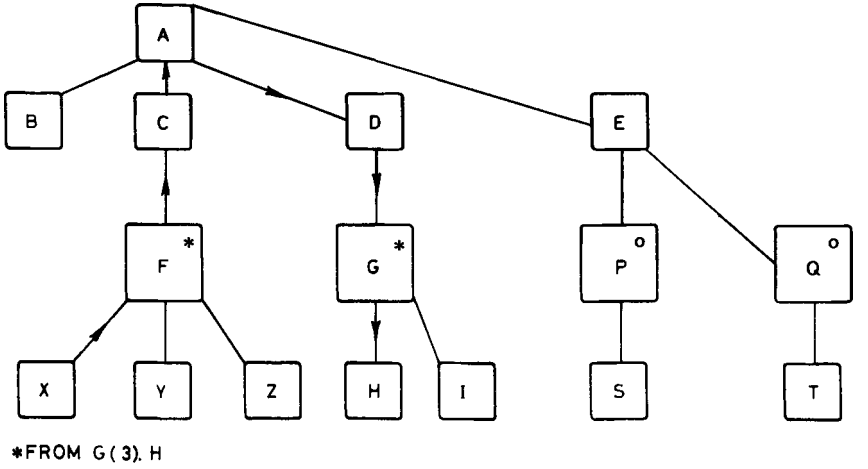


Fig. 11 Name references in RADS

a value of the same type, i.e. with the same structure and data characteristics, as the object node.

Note that we prefix names using the point as separator for qualification, a.b.c. being the *c* element that is a component of *b* which is a component of *a*. Subscripts are written in parentheses following the iterate to which they apply.

5.2.6 Module invocation: A module can be invoked from within another module. We give as an example a module that takes two numbers and calls the square-root module to determine the root of the sum of their squares.

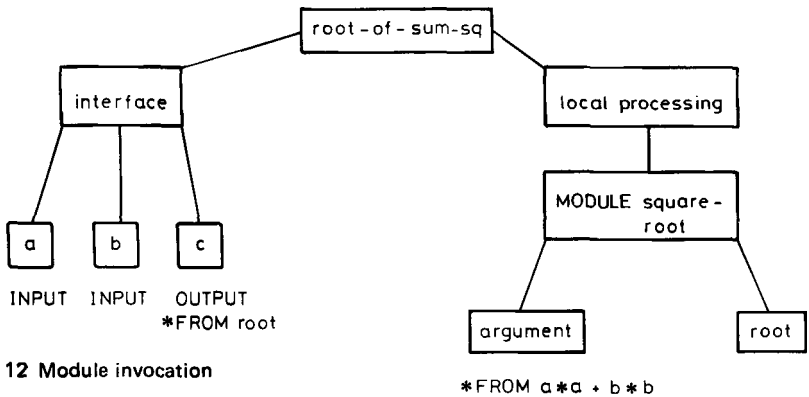


Fig. 12 Module invocation

Note that only the interface of the module 'square-root' is visible to this module.

5.3 Updating in RADS

5.3.1 The basic update model: Updating is the replacement of a value of a node with a new value for that node. A simple example and a more complex one will illustrate this.

Fig. 13 shows a module 'half' that has an item on its interface declared as UPDATE. The module supplies a new value for the item.

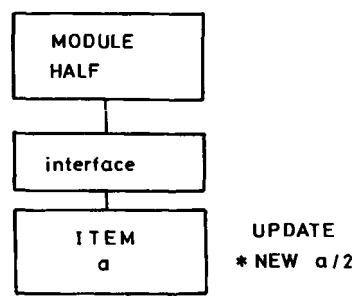


Fig. 13 Module half

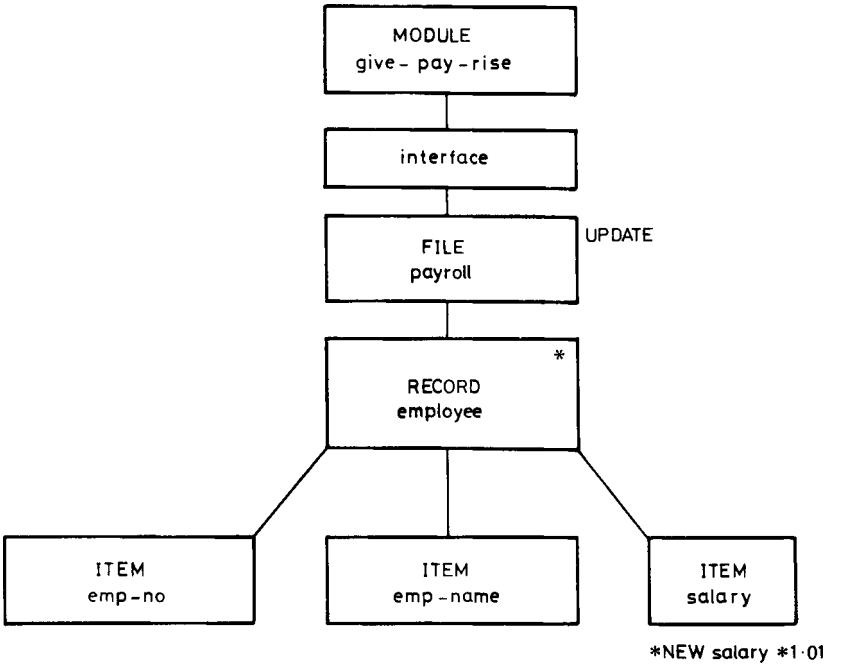


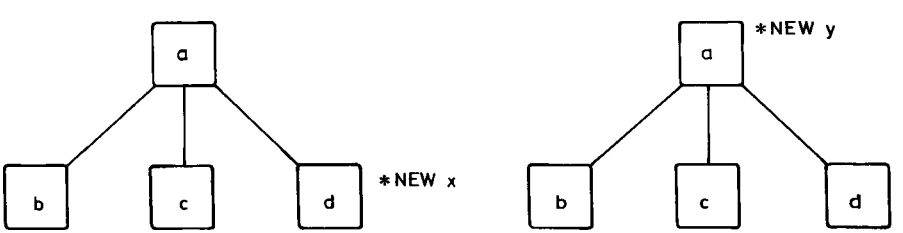
Fig. 14 Module to update the payroll file

Consider now a somewhat more complex example. The UPDATE object on the interface of the module 'give-pay-rise' is a whole payroll file.

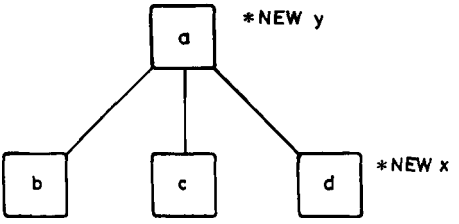
It consists of many employee records. Each employee record contains several items including one called 'salary'. The module is to give all the employees a 1% pay rise, and has the above structure (Fig. 14).

In this example the *NEW property assigns new values to all the 'salary' items in the file. The programmer can view the entire file update as one operation, as a single change of state.

A new value may be specified (by a *NEW property) for a terminal node (always a single data item) or for a compound data object. Either of the following is permitted:



A new value for a node may not be specified in more than one way.
Thus the following is *not* permitted as the new value for *d* is described twice:



Thus RADS allows state changes to be arbitrarily large. Usually, the larger the state change, the easier it is to think about the problem. Hence, the RADS programmer will tend to write programs with as few state changes as possible. However, although not theoretically necessary, it is advantageous from a usability point of view to be able to model certain state changes within the module.

5.3.2 Variables and views: Thus far we have discussed nodes in a structure that may have at most one old value and at most one new value. We term such nodes 'transient'.

We now introduce the concept of the 'variable'. A variable is any node that undergoes a succession of state changes, each one replacing its value with an updated value. A variable that is a single item is similar to a variable in a procedural language, except that it has no resource allocation semantics. However, a variable in RADS may be an arbitrarily large object, for example, a file or a database.

Updating of variables is the RADS equivalent to assignment in procedural languages. The value of a variable changes a number of times so that a reference to it retrieves different values from different nodes in the process structure. RADS differs from procedural languages, and avoids their lack of clarity by only allowing such assignments in a very controlled way, and in grouping together a collection of individual changes to different items and executing them simultaneously. The usefulness of this was expounded by Gries and by Floyd,[9,13] who favour a multiple assignment statement in procedural languages.

The variable is special in RADS as it is the only node in a process structure that may take more than two values. We provide a special mechanism to control value changes. A node in a structure may be specified to be a 'view' of another node. The node that is the view must be derived from the 'viewed' node. Some, but not necessarily all of the item nodes in the view will match, or correspond to, nodes in the viewed node. Those that are to be updated must match in a one-to-one relationship.

A view is generally a transient whose nodes may only have an old and a single new value. If a view is a variable it will have a succession of intermediate values resulting in a final 'new' value. The new value of a matching node in a view will be reflected back to update the viewed node.

The following two diagrams show a module to update a file of customers, using a transaction file.

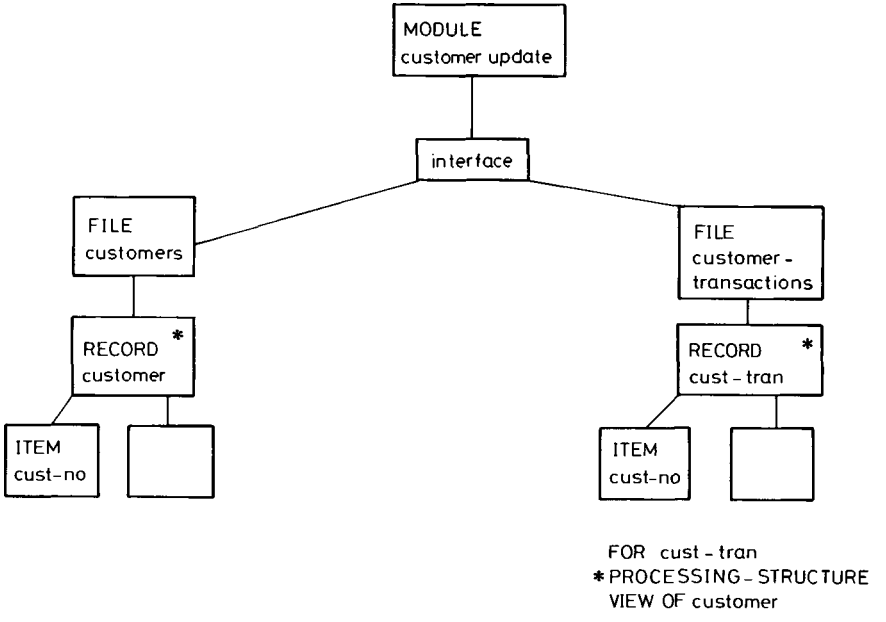


Fig. 15 Data used by the customers update module

Fig. 15 shows the initial structure of the module. Its interface consists of the two files. We need to specify the processing to be done for each cust-tran record. The *PROCESSING-STRUCTURE property says that a view of the customer record will be 'added' to the program's control structure, giving that shown in Fig. 16

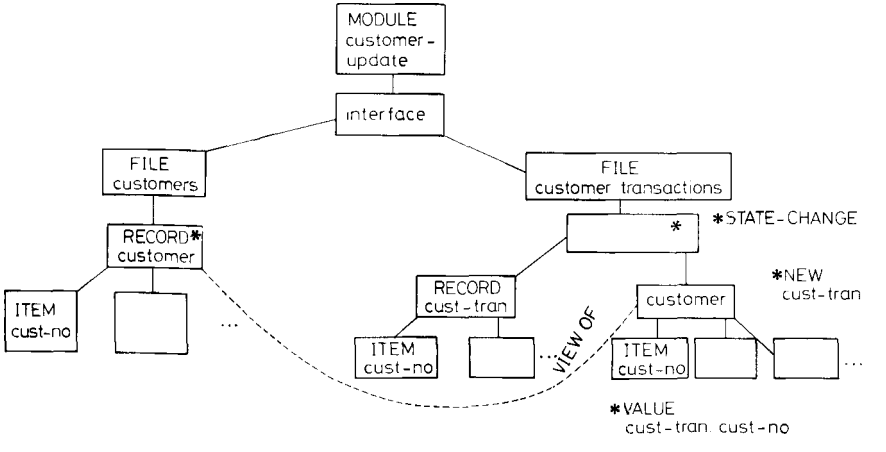


Fig. 16 Complete control structure for the customer-update module

*PROCESSING-STRUCTURE is one of a number of properties available to a RADS programmer to alter a program's structure. We see here a good example of the duality between data structure and process structure in RADS.

The view of 'customer' inherits the structure of customer. The *VALUE property asserts that the customer number for the viewed customer must have the specified value. The *NEW property gives the new value to the viewed node, which is reflected back to the customer record in the customer's file, updating it, for each transaction.

5.3.3 State changes: If a node is designated a state change, then all the components of a state change node that have new values specified appear to get new values at the same time. If a node is a view, the new values of the view are reflected into the source variable when the view changes state.

A state change node may be a component of another state change node. If this is the case the component state change is considered to occur before the outer state change.

References to nodes from below a common state change all refer to old values. References from a node to another node, such that the path (see Fig. 11) passes down to another node that is a state change node, retrieve the new value of the referenced node.

As an example, consider the module shown in Fig. 17. The values of a, b, c are permuted. (The new value of *a* is 2, and so on). The *NEW properties, refer to the old values.

The *FROM on R, however, has a reference that passes through the explicit state change, and thus references the new value, i.e. 2.

If a variable is subordinate to a state change all update views of the variable must be defined within the scope of the same state change since a view from outside the

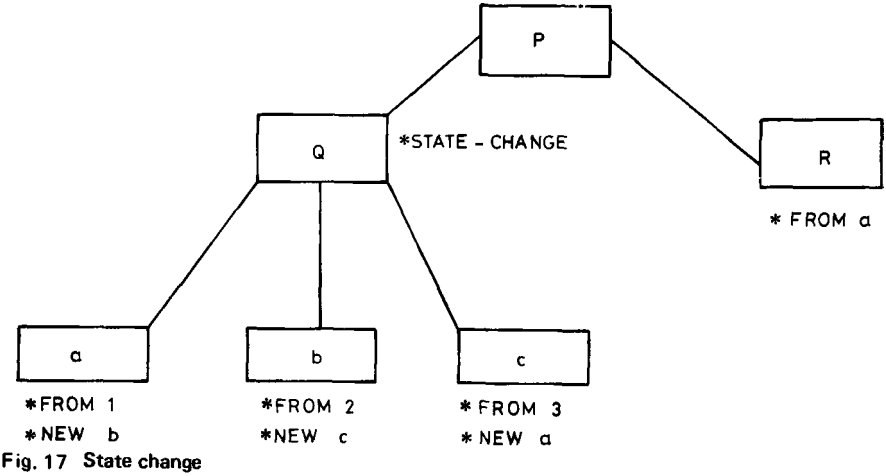


Fig. 17 State change

state change must retrieve the single final new value. Hence from outside the owning state change of a variable the variable appears to be a transient.

As mentioned in Section 5.3.2, a major motivation for the introduction of state changes is to provide a model for variables. In particular, nodes that are views will normally also be state change nodes.

5.3.4 Simultaneous equations example: To show the generality of the language a complete program is included to generate the solution to a set of simultaneous equations.

Fig. 18 shows the generated process structure required to solve the equations which are represented by the problem matrix.

The only new feature here is the implied integer iterate.index which is the iterate number of a given iterate within the iteration. Its semantics are the exact equivalent of *FIRST 1 *NEXT index+1.

The solution involves an iteration of two views. The first view selects a pivotal row, being the first row that has a non-zero coefficient of the next variable. The row is divided by this coefficient.

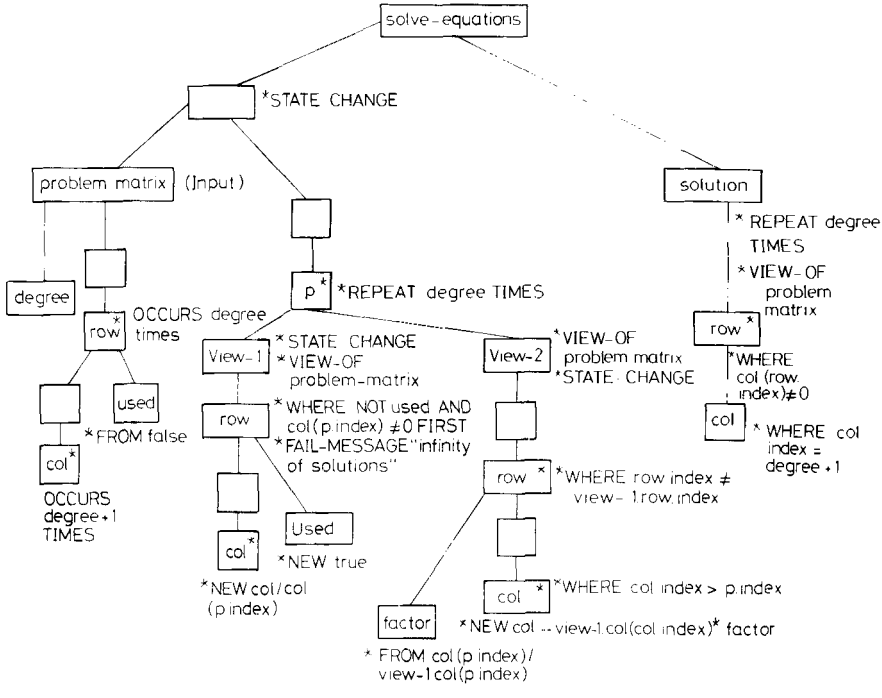


Fig. 18 Solution of simultaneous equations

The second view selects all the other rows and subtracts an appropriate multiple of the pivotal row from each so as to leave zero coefficients of the pivotal variable.

The p iteration repeats until all variables have been pivoted. The matrix is then a left-hand square component with a single 1 in each row and in each column and a right-hand column which represents the solution.

The solution is the iteration solution.col, where the values are selected in the correct order.

At view-1.row the structure may fail if, for example, the matrix has two identical rows. A failure message is generated which, in this example, would become the diagnostic for the program.

Note that in both views all the new values could be calculated simultaneously.

6 Summary

The main objective of RADS is to improve programmer productivity to meet the requirements of the eighties. It is hoped that this can be achieved using a new process model based upon the generation of Jackson structures.

The RADS process model differs fundamentally from the conventional von Neumann machine which underlies procedural programming languages. In the RADS machine the process proceeds in far fewer state changes, in each of which many new values may be computed.

Developing a RADS program requires a complete definition of the process structure according to structured programming rules. This enables the process to be defined interactively and incrementally and to be easily represented in diagram form.

There is no concept of input or output commands as all data, including a database, may be considered as part of the process itself. The process model imposes more problems for the compiler, as it has to turn the non-procedural definition into a procedural program and to analyse data requirements to determine the need for memory and transfers of external data throughout the process. However, it is possible that a compiler could do more optimisation as the problem statement is more general. It is also less dependent upon hardware architecture and we can easily envisage generation of code for parallel processing machines.

Experience with the general model leads us to believe that its use to describe easy problems is straightforward. The model assists the solution of intellectually challenging problems by forcing the structuring of the problem. Coding the solution is more straightforward than in COBOL or Fortran and is less error prone because more of the problem is visible at any time. Also, we expect that most mistakes can be corrected interactively and the compiler will be able to find errors that would otherwise cause obscure faults at run-time.

The low level model is not the complete answer to programmer productivity. Higher-level problem-oriented facilities are very important, and our experience with file processing and reporting programs, which use many of the RADS high level functions, show impressive productivity gains. In general, as more high level functions can be used, so the productivity gain increases.

RADS high level functions are defined in terms of lower level functions. This not only ensures a rigorous definition, but also means that high level functions are easy to explain, and easy to supplement with lower level facilities where required.

References

1 ICL Technical Publication 6504 'Data Dictionary System' published by ICL in 1980.
2 BACKUS, J.: 'Can programming be liberated from the von Neumann style?', Communications of the ACM August 1978, **21** (8).
3 RUNYAN, L. and SCHATZ, W.: 'Applications development', *Datamation* **27** (3), March 1981.
4 US Department of Defence, 'Requirements for Ada Programming Support Environments - 'Stoneman' ' Feb. 1980.
5 Reference Manual for the Ada Programming Language, United States Department of Defence, July 1980.
6 JACKSON, M.A.: *'Principles of program design'*, Academic Press.
7 Draft Proposed Revised X3.23 American National Standard Programming Language COBOL, published by the ANSI Technical Committee X3J4 in May 1981.
8 ASHCROFT, E.A. and WADGE, W.W.: 'LUCID, a non-procedural language with iteration', Communications of the ACM, July 1977, Vol. 20, No. 7.
9 FLOYD, R.W.: 'The paradigms of programming', Communications of the ACM, August 1979, **22** (8), pp. 455-460.
0 HUGHES, J.W.: 'A formalization and explication of the Michael Jackson method of program design', *Software – Practice and Experience* 1979, 9, pp. 191-202.
1 BROWN, A.P.G., COSH, H.G. and GRADWELL, D.J.L.: 'Database processing in RADS' in Proceedings of the First British National Conference on Databases, July 1981.
2 DAHL, O-J, DIJKSTRA, E.W. and HOARE, C.A.R.: *'Structured programming'* Academic Press, London (1972).
3 GRIES, D.: 'Educating the programmer : notation, proofs and the development of programs', *Information Processing* 1980.

A moving-mesh plasma equilibrium problem on the ICL Distributed Array Processor

P.Kirby

Culham Laboratory, Abingdon, Oxon OX14 3DB
(Euratom/UKAEA Fusion Association)

Abstract

A description is given of a moving-mesh plasma equilibrium problem that has been coded in DAP Fortran and run on the ICL DAP at Queen Mary College. The problem provides an interesting example for the DAP, because the calculation requires a range of logical and numerical operations, in particular the respacing of the mesh as the computation proceeds. The DAP code ran 29 times faster than a similar (but not identical) conventional Fortran code on an ICL 2976.

1 Introduction

This paper describes a moving-mesh plasma equilibrium problem that has been run on the Distributed Array Processor (DAP) at Queen Mary College. This work, part of a continuing assessment of the DAP, was done to demonstrate the solution of a substantial problem on the production machine. The equilibrium problem to be described provides an interesting example, since it requires a range of computational operations, in particular the re-spacing of the mesh as the calculation proceeds. We show that all the necessary operations may be done by parallel algorithms and may be programmed easily and naturally in DAP Fortran.

2 Background

The equilibrium equation of plasma physics is the mildly non-linear equation 1.[1]

$$R\frac{\partial}{\partial R}\left(\frac{1}{R}\frac{\partial \chi}{\partial R}\right)+\frac{\partial^2 \chi}{\partial Z^2}=-FF_\chi-\mu_0 R^2 p_\chi \qquad (1)$$

With suitable boundary conditions, this describes the equilibrium of an axisymmetric toroidal plasma. R and Z are cylindrical coordinates based on the major radius of the torus (Fig. 1). $2\pi\chi$ is the poloidal flux of magnetic field. $F(\chi)$ and $p(\chi)$ are arbitrary functions of χ. (F/R is the toroidal magnetic field and p is the plasma pressure.)

The usual approach to the numerical solution of eqn. 1 is to difference the operator on a uniform R, Z mesh and to solve the equations to obtain $\chi(R_i, Z_j)$.[2] The level surfaces of χ (the 'χ surfaces' or 'magnetic surfaces') are then found by contouring

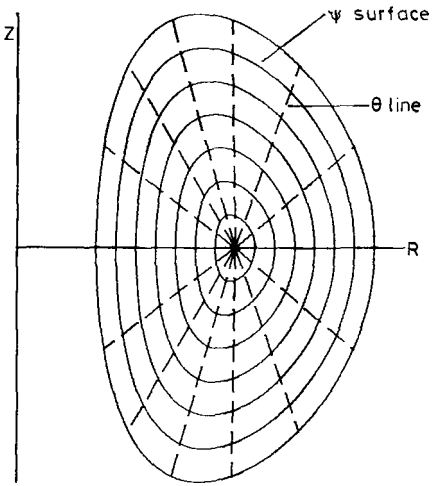


Fig. 1 Definition of coordinates

the array of χ values. In cases of interest, these surfaces are closed and nested. We shall think of χ as increasing monotonically outwards from its minimum value (the magnetic axis).

Another approach, in which the χ surfaces are found as an integral part of the equilibrium calculation, is to solve eqn. 1 in non-orthogonal 'magnetic surface' coordinates ψ, θ. ψ is a radial coordinate with closed, nested surfaces and θ is an arbitrary angle-like coordinate (Fig. 1). The initial ψ, θ mesh $(R(\theta_i, \psi_j), Z(\theta_i, \psi_j))$ is arbitrary. The ψ mesh is then adjusted until χ is made independent of θ to a specified accuracy i.e. $\chi = \chi(\psi)$. The χ surfaces are then the known ψ surfaces.
In addition to avoiding the errors involved in contouring an array of χ values, this approach is unaffected by the existence of steep gradients and closely spaced χ surfaces in real space. Also, the implementation of the common boundary condition χ = const. is easy because the boundary is then a ψ surface.

A general procedure for the solution of the equilibrium equation in ψ, θ coordinates was given in Delucia *et al.*[3] The steps are as follows:

(1) Set up a finite difference approximation to the equilibrium equation in ψ, θ coordinates.
(2) Set up the shape of the boundary.
(3) Using an arbitrary position $(R = R_0, Z = 0)$ for the axis of the ψ, θ coordinates, set up an initial ψ, θ mesh i.e.

$$R(\theta_i, \psi_j), Z(\theta_i, \psi_j)$$

(4) Solve the equilibrium equation on the initial mesh. Locate the magnetic axis (the point of minimum χ) and set up a new initial ψ, θ mesh using the magnetic axis as the position of the coordinate axis. The coordinate axis is now fixed for the rest of the calculation. (The rest of the calculation

consists of the radial adjustment of the mesh.)

(5) Solve the equilibrium equation on the new mesh and determine whether $\chi = \chi(\psi)$ to the required accuracy.

(6) If $\chi \neq \chi(\psi)$, interpolate the ψ mesh along the θ lines so that $\chi = \chi(\psi)$ holds.

(7) Repeat steps (5) and (6) until the solution of the equilibrium equation satisfies $\chi = \chi(\psi)$ to the required accuracy. The solution of the problem is then complete.

It is the implementation on the DAP of this strategy that is described here.

3 Definition of the present problem

The magnetic surface equlibrium problem to be solved is set out below:

(1) Fixed boundary calculation in JET (Joint European Torus) geometry.

(2) Dirichlet boundary condition $\chi = 0$.

(3) The toroidal current in the equilibrium is specified.

(4) ψ, θ have the domain

$$0 \leq \psi \leq 1$$

$$0 \leq \theta \leq 2\pi$$

$\psi = 1$ is the boundary. $\psi = 0$ is the coordinate axis.
θ = const. are straight lines ('spokes') from the coordinate axis to the (fixed) boundary points.
ψ is to be equally spaced in R outwards from the coordinate axis along $Z = 0$.

(5) The system has up/down symmetry and is solved in the upper half-plane, $Z \geq 0$ $(0 \leq \theta \leq \pi)$.

(6) The expressions for p and F are (in dimensional form)

$$F(\chi) = R_0 B_0 \, (1 - \gamma \, (-\chi/B_0)^{\beta})$$
$$p(\chi) = p_0 \, (-\chi/B_0)^{\alpha} \qquad (2)$$

$R_0, B_0, p_0, \alpha, \beta$ are parameters and γ is adjusted during the calculation in order to keep the toroidal current fixed.

(7) The magnetic field, plasma pressure and current are normalised according to

$$\underline{B} = B_0 \, \underline{\tilde{B}}$$
$$p = (B_0{}^2/\mu_0)\,\tilde{p} \qquad (3)$$
$$\underline{j} = (B_0/\mu_0)\,\underline{\tilde{j}}$$

(8) The solution of the non-linear equations is by SOR and Picard linearisation. An integral relation is applied at the coordinate axis.

4 Description of the calculation and implementation on the DAP

4.1 Mapping of the problem onto the DAP

The layout of the ψ, θ mesh in real space is shown in Fig. 1 and its mapping onto the DAP is shown in Fig. 2. For simplicity, the problem has been mapped onto one 64 x 64 plane. There would be no difficulty in principle in using a mesh $64n$ x $64m$ with $n, m > 1$. The rays $\theta = -\Delta\theta$ and $\theta = \pi + \Delta\theta$ are needed to implement up/down symmetry. The mesh is defined by

$$\Delta\psi = 1/63$$

$$\Delta\theta = \pi/61$$

$$\psi = (j - 1)\,\Delta\psi \qquad 1 \leq j \leq 64$$

$$\theta = (i - 2)\,\Delta\theta \qquad 1 \leq i \leq 64$$

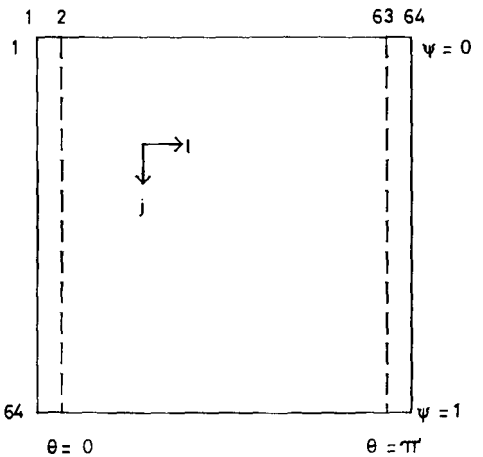


Fig. 2 Mapping onto the DAP

4.2 The finite difference equations ($\psi > 0$)

The equilibrium equation in ψ, θ coordinates may be obtained by direct transformation of eqn. 1. Alternatively, it may be obtained from first principles using tensor analysis (Appendix 1). The expression is

$$\frac{\partial}{\partial\psi}\left(h_{\psi\psi}\,\chi_{\psi} + h_{\theta\psi}\,\chi_{\theta}\right) + \frac{\partial}{\partial\theta}\left(h_{\theta\psi}\,\chi_{\psi} + h_{\theta\theta}\,\chi_{\theta}\right) = -J\left(\frac{FF_{\chi}}{R^2} + p_{\chi}\right) \qquad (4)$$

The metric coefficients $h_{\alpha\beta}$ are given by

$$h_{\psi\psi} = (R_\theta{}^2 + Z_\theta{}^2)/J$$

$$h_{\theta\theta} = (R_\psi{}^2 + Z_\psi{}^2)/J \qquad (5)$$

$$h_{\theta\psi} = -(R_\psi R_\theta + Z_\psi Z_\theta)/J$$

and the Jacobian J by

$$J = R(R_\psi Z_\theta - R_\theta Z_\psi) \ . \qquad (6)$$

Here, $\chi_\psi \equiv \partial\chi/\partial\psi, R_\psi \equiv \partial R/\partial\psi$ etc.

At the coordinate axis, J vanishes and so a special treatment is needed. For $\psi > 0$ following Delucia *et al*,[3] we use the following centered difference schemes:

$$\left[\frac{\partial}{\partial\psi}(h_{\psi\psi}\,\chi_\psi)\right]_{i,j} = \left((h_{\psi\psi}\,\chi_\psi)_{i,j+\frac{1}{2}} - (h_{\psi\psi}\,\chi_\psi)_{i,j-\frac{1}{2}}\right)/\Delta\psi$$

$$(\chi_\psi)_{i,j} = (\chi_{i,j+\frac{1}{2}} - \chi_{i,j-\frac{1}{2}})/\Delta\psi$$

$$\left[\frac{\partial}{\partial\theta}(h_{\theta\theta}\,\chi_\theta)\right]_{i,j} = \left((h_{\theta\theta}\,\chi_\theta)_{i+\frac{1}{2},j} - (h_{\theta\theta}\,\chi_\theta)_{i-\frac{1}{2},j}\right)/\Delta\theta$$

$$(\chi_\theta)_{i,j} = (\chi_{i+\frac{1}{2},j} - \chi_{i-\frac{1}{2},j})/\Delta\theta$$

$$\left[\frac{\partial}{\partial\psi}(h_{\theta\psi}\,\chi_\theta)\right]_{i,j} = \left((h_{\theta\psi}\,\chi_\theta)_{i,j+\frac{1}{2}} - (h_{\theta\psi}\,\chi_\theta)_{i,j-\frac{1}{2}}\right)/\Delta\psi$$

$$(\chi_\theta)_{i,j+\frac{1}{2}} = \left((\chi_{i+1,j+1} - \chi_{i-1,j+1}) - (\chi_{i+1,j} - \chi_{i-1,j})\right)/4\Delta\theta$$

$$\left[\frac{\partial}{\partial\theta}(h_{\theta\psi}\,\chi_\psi)\right]_{i,j} = \left((h_{\theta\psi}\,\chi_\psi)_{i+\frac{1}{2},j} - (h_{\theta\psi}\,\chi_\psi)_{i-\frac{1}{2},j}\right)/\Delta\theta$$

$$(\chi_\psi)_{i+\frac{1}{2},j} = \left((\chi_{i+1,j+1} - \chi_{i+1,j-1}) + (\chi_{i,j+1} - \chi_{i,j-1})\right)/4\Delta\psi$$

We define the coefficients $H_{\alpha\beta}$

$$H_{\psi\psi} = h_{\psi\psi} / (\Delta\psi)^2$$

$$H_{\theta\theta} = h_{\theta\theta} / (\Delta\theta)^2 \qquad (7)$$

$$H_{\theta\psi} = h_{\theta\psi} / 4\,\Delta\theta\,\Delta\psi$$

and the coefficients $(a_n)_{i,j}$ by

$$(a_0)_{i,j} = (H_{\psi\psi})_{i,j+\frac{1}{2}} + (H_{\psi\psi})_{i,j-\frac{1}{2}} + (H_{\theta\theta})_{i+\frac{1}{2},j} + (H_{\theta\theta})_{i-\frac{1}{2},j}$$

$$(a_1)_{i,j} = (H_{\theta\psi})_{i,j-\frac{1}{2}} + (H_{\theta\psi})_{i-\frac{1}{2},j}$$

$$(a_2)_{i,j} = (H_{\psi\psi})_{i,j-\frac{1}{2}} + (H_{\theta\psi})_{i-\frac{1}{2},j} - (H_{\theta\psi})_{i+\frac{1}{2},j}$$

$$(a_3)_{i,j} = -(H_{\theta\psi})_{i+\frac{1}{2},j} - (H_{\theta\psi})_{i,j-\frac{1}{2}}$$

$$(a_4)_{i,j} = (H_{\theta\theta})_{i-\frac{1}{2},j} - (H_{\theta\psi})_{i,j+\frac{1}{2}} + (H_{\theta\psi})_{i,j-\frac{1}{2}}$$

$$(a_5)_{i,j} = (H_{\theta\theta})_{i+\frac{1}{2},j} - (H_{\theta\psi})_{i,j-\frac{1}{2}} + (H_{\theta\psi})_{i,j+\frac{1}{2}}$$

$$(a_6)_{i,j} = -(H_{\theta\psi})_{i,j+\frac{1}{2}} - (H_{\theta\psi})_{i-\frac{1}{2},j}$$

$$(a_7)_{i,j} = (H_{\psi\psi})_{i,j+\frac{1}{2}} - (H_{\theta\psi})_{i-\frac{1}{2},j} + (H_{\theta\psi})_{i+\frac{1}{2},j}$$

$$(a_8)_{i,j} = (H_{\theta\psi})_{i,j+\frac{1}{2}} + (H_{\theta\psi})_{i+\frac{1}{2},j}$$

Then, the finite difference representation of eqn. 4 may be written

$$\begin{aligned}(a_0)_{i,j}\chi_{i,j} &- (a_1)_{i,j}\chi_{i-1,j-1} - (a_2)_{i,j}\chi_{i,j-1} \\ &- (a_3)_{i,j}\chi_{i+1,j-1} - (a_4)_{i,j}\chi_{i-1,j} \\ &- (a_5)_{i,j}\chi_{i+1,j} - (a_6)_{i,j}\chi_{i-1,j+1} \\ &- (a_7)_{i,j}\chi_{i,j+1} - (a_8)_{i,j}\chi_{i+1,j+1} \\ &= B_{i,j}\end{aligned} \qquad (8)$$

in which

$$B_{i,j} = \left[J\left(\frac{FF_\chi}{R^2} + p_\chi\right)\right]_{i,j} \qquad (9)$$

In matrix form eqn. 8 may be written as

$$A\chi = B(\chi) \tag{10}$$

in which A is weakly diagonally dominant.

In order to compute the coefficients $H_{\alpha\beta}$ in eqn. 7 and the Jacobian in eqn. 9, the derivatives R_θ, Z_θ, R_ψ, Z_ψ are needed at mesh points and midpoints. We use the centered difference schemes

$$\begin{aligned}
(R_\theta)_{i,j} &= (R_{i+1,j} - R_{i-1,j}) / 2\Delta\theta \\
(R_\theta)_{i+\frac{1}{2},j} &= (R_{i+1,j} - R_{i,j}) / \Delta\theta \\
(R_\theta)_{i,j+\frac{1}{2}} &= \left((R_{i+1,j+1} - R_{i-1,j+1}) + (R_{i+1,j} - R_{i-1,j}) \right) / 4\Delta\theta
\end{aligned} \tag{11}$$

and similarly for the other derivatives.

In DAP Fortran, the differences are easy to evaluate using nearest neighbour shifts. Consider, for example, the computation of eqn. 11. This may be written

```
RT_A(,) = (R(,+) -R) *MAT(RDT)
W(,)    = (R(,+) - R(,-))*0-5E0*MAT(RDT)
RT_B(,) = (W + W(+,))*0-5E0
RT(,)   = W(,)
```

Here, the postfixes _A and _B refer to the midpoints $(i+\frac{1}{2},j)$ and $(i,j+\frac{1}{2})$, respectively.

4.3 The finite difference equations ($\psi = 0$)

The value of χ at the axis of the coordinates is obtained using an integral relation. Integrating eqn. 4 over the area enclosed by the surface $\psi = \Delta\psi/2$ ($j = 3/2$) and using periodicity in θ, we obtain

$$\int_0^{2\pi} (h_{\psi\psi}\,\chi_\psi + h_{\theta\psi}\,\chi_\theta) \Big|_0^{\Delta\psi/2} \mathrm{d}\theta = -\int_0^{\Delta\psi/2}\int_0^{2\pi} \left(\frac{FF_\chi}{R^2} + p_\chi \right) J\,\mathrm{d}\psi\,\mathrm{d}\theta \tag{12}$$

The lower limit on the left hand side of eqn. 12 gives no contribution, since the derivatives with respect to θ vanish at the origin. Using the centered difference schemes

$$(\chi_\psi)_{i,3/2} = (\chi_{i,2} - \chi_{i,1})/\Delta\psi$$

$$(\chi_\theta)_{i,3/2} = \tfrac{1}{2}\left((\chi_\theta)_{i,2} + (\chi_\theta)_{i,1}\right) = \tfrac{1}{2}(\chi_\theta)_{i,2}$$

$$= (\chi_{i+1,2} - \chi_{i,-1,2})/4\Delta\theta$$

and the trapezoidal rule to approximate the integrals, eqn. 12 becomes

$$\chi_{i,1} \sum_{i=2}^{63} \epsilon(H_{\psi\psi})_{i,3/2} = \sum_{i=2}^{63} \epsilon(H_{\psi\psi})_{i,3/2}\,\chi_{i,2}$$

$$+ \sum_{i=2}^{63} \epsilon(H_{\theta\psi})_{i,3/2}(\chi_{i+1,2} - \chi_{i-1,2}) \qquad (13)$$

$$+ \sum_{i=2}^{63} \epsilon\left(\frac{FF_\chi}{R^2} + p_\chi\right)_{i,5/4} J_{i,5/4}$$

in which $\epsilon = \tfrac{1}{2}$ for $i = 2,63$ and 1 otherwise. (Since the last two terms on the right hand side of eqn 13 are very small, it would probably be sufficiently accurate to omit them and therefore compute $\chi_{i,1}$ as an average of the $\chi_{i,2}$.)

In DAP Fortran, the sums are obtained using the built-in function SUM. For example, the sum on the left hand side of eqn. 13 is evaluated by

```
SUM1_1D = SUM(STEP_FUNCTION()*HPP_D(2,))
```

4.4 Definition of the boundary shape

The boundary is defined by

$$R_{i,64} = R_0 + a\cos(\theta_i + \delta\sin\theta_i)$$

$$Z_{i,64} = \epsilon a \sin\theta_i$$

$$\theta_i = (i-2)\frac{\pi}{61} \qquad 1 \le i \le 64$$

in which R_0, a, δ, ϵ are parameters. For the Dee shaped cross-section in JET,

$$R_0 = 2{\cdot}96, \quad a = 1{\cdot}25, \quad \delta = 0{\cdot}2539, \quad \epsilon = 1{\cdot}68$$

In DAP Fortran,

```
R(64,) = VEC(RZERO)+VEC(ABOUND)*COS(THETA+VEC(DBOUND)
                                    *SIN(THETA))

Z(64,) = VEC(EBOUND)*VEC(ABOUND)*SIN(THETA)
```

These points are fixed throughout the calculation.

4.5 *Initial guess for the ψ, θ mesh*

Let R_c be the required position of the coordinate axis. At the start of the calculation, we make the guess $R_c = R_0$. When the magnetic axis (R_m) has been found, R_c is changed to R_m and then remains fixed for the remainder of the calculation.

The θ lines are defined to be straight lines joining R_c to the fixed boundary points set up in Section 4.4. The ψ surfaces are then equally spaced in distance along the θ lines, between the coordinate axis and the boundary. The mesh values are therefore given by

$$R_{i,j} = R_c + \frac{(j-1)}{63}\,(R_{i,64} - R_c)$$

$$Z_{i,j} = \frac{j-1}{63} Z_{i,64} \qquad \begin{matrix} 1 \leq j \leq 63 \\ 1 \leq i \leq 64 \end{matrix} \tag{14}$$

As an example, in DAP Fortran, the expression in eqn. 14 for $Z_{i,j}$ is

```
Z(ROWS(1,63)) = MATC(STEP)*MATR(Z(64,)).
```

Here, STEP is a vector containing 0(1/63)1. MATC(MATR) are built-in DAP Fortran functions that produce matrices with each column (row) set equal to their argument.

4.6 *The initial guess for χ*

The initial guess for χ is based on the value of χ at the surface of a cylinder of length $2\pi R_0$ carrying the same current. Consider quantities normalised according to eqn. 3. If j is a measure of the current density, the total current I is measured by

$$I \sim \pi a^2 j$$

and the poloidal field B_θ is measured by

$$B_\theta(r) \sim rj$$

The poloidal flux (χ_{cyl}) in the cylinder is therefore given by

$$2\pi\chi_{cyl} = 2\pi R_0 \int_0^a B_\theta(r)\,dr \sim R_0 I$$

The initial value of χ at the coordinate axis is then taken as

$$\chi_{i,1} = -R_0 I/2\pi \qquad (1 \leq i \leq 64)$$

χ on the rest of the mesh is specified to vary linearly from the axis value to zero on the boundary (the boundary condition):

$$\chi_{i,j} = \frac{64-j}{63}\chi_{i,1}$$

In DAP Fortran,

```
CHI(,) = - MATC(STEP( )*VEC(CHI_CYLINDER))
```

in which STEP contains 1(-1/63)0.

4.7 *Solution of the non-linear equations*

The eqns. 8 are solved using SOR and Picard linearisation. In matrix form, this scheme may be written

$$\chi^{n+1} = (1-\omega)\chi^n + \omega D^{-1}\,(L\chi^{n+1} + U\chi^n + B(\chi^n)) \qquad (15)$$

in which $A = D-U-L$ and n is the iteration number.

To implement the nine point scheme in eqn. 15 on the DAP, we take a '4 colour' ordering of the mesh points, as shown in the diagram:

```
   |   |   |
 - b - w - b -        (b = 'black')
   |   |   |
 - r - l - r -        (w = 'white')
   |   |   |
 - b - w - b -        (r = 'red')
   |   |   |
                      (l = 'blue')
```

Then, each mesh point of a given colour is connected by eqn. 8 only to mesh points of a different colour. In turn, the black, white, red and blue values may therefore be updated simultaneously, at each stage using the latest values at the other mesh points. With this ordering, the matrix A in eqn. 10 has diagonal blocks that are diagonal but is not 2-cyclic.

In DAP Fortran, an SOR iteration (including up/down symmetry) may be written

```
CHI(BLACK) = OMEGA_C*CHI + OMEGA_OVER_A0*(
                                  A1*CHI(-,-) +
                                  A2*CHI(-,)  +
                                  A3*CHI(-,+) +
                                  A4*CHI(,-)  +
                                  A5*CHI(,+)  +
                                  A6*CHI(+,-) +
                                  A7*CHI(+,)  +
                                  A8*CHI(+,+) +
                                  B)

CHI(,64)      = CHI(,62)
CHI(WHITE)    = < ditto >
CHI(,1)       = CHI(,3)
CHI(RED)      = < ditto >
CHI(,64)      = CHI(,62)
CHI(BLUE)     = < ditto >
CHI(,1)       = CHI(,3)
```

in which BLACK, WHITE, RED, BLUE are logical matrices corrsponding to the 4 colour ordering. For example,

```
BLACK(,) = ALTC(1) .AND. ALTR(1)
```

With no constraints, if eqn. 15 converges, it converges to the trivial solution $\chi = 0$ everywhere. To prevent this, we impose the constraint that the toroidal current is fixed. After each iteration, the toroidal current $I(\gamma)$ is computed in terms of the parameter γ in the function F. The value of γ to be used for the next iteration is then obtained from $I(\gamma) = I = \text{const}$.

The expression for the toroidal current is

$$I = -\int_0^1 \int_0^{2\pi} \left(\frac{FF_\chi}{R^2} + p_\chi \right) J \mathrm{d}\psi \, \mathrm{d}\theta$$

Using eqn. 2 for F, this may be written

$$I = a\gamma^2 + b\gamma + c \tag{16}$$

in which

$$a \equiv \int_0^1 \int_0^{2\pi} \frac{{R_0}^2 \beta}{R^2} (-\chi)^{2\beta - 1} J \mathrm{d}\psi \, \mathrm{d}\theta$$

$$b \equiv -\int_0^1 \int_0^{2\pi} \frac{{R_0}^2 \beta}{R^2} (-\chi)^{\beta-1} J \mathrm{d}\psi \, \mathrm{d}\theta \qquad (17)$$

$$c \equiv -\int_0^1 \int_0^{2\pi} p_\chi J \, \mathrm{d}\psi \, \mathrm{d}\theta$$

If an iteration makes the current change from the specified value, γ is adjusted to restore its value; γ is given the value of the smaller root (in absolute value) of eqn. 16. (The choice of the smaller root is arbitrary.) If an iteration makes the current increase above a certain value, it may not be possible to use γ to reduce it to the required value. In this case, γ is chosen to give the smallest current possible. This occurs for $\gamma = - b/2a$ and corresponds to setting the discriminant of eqn. 16 equal to zero.

In general form, the integrals in eqn. 17 may be written as

$$v = \int_0^1 \int_0^{2\pi} Q(\psi,\theta) J \, \mathrm{d}\psi \, \mathrm{d}\theta$$

To evaluate this, we first compute an average value of Q at the centre of each element of area:

$$\bar{Q}_{i,j} = \tfrac{1}{4} \left(Q_{i+\frac{1}{2},j} + Q_{i-\frac{1}{2},j} + Q_{i+\frac{1}{2},j-1} + Q_{i-\frac{1}{2},j-1} \right)$$

$$Q_{i-\frac{1}{2},j} = \tfrac{1}{2} \left(Q_{i-1,j} + Q_{i,j} \right)$$

and then approximate the integral by

$$v \cong 2 \, \Delta\psi \, \Delta\theta \sum_{i=2}^{63} \sum_{j=2}^{64} \epsilon \bar{Q}_{i,j} J_{i,j-\frac{1}{2}}$$

In DAP Fortran, the integral may be written

```
W(,)     =Q(,-)+Q(,+)+2.0E0*Q(,)

QBAR(,) = 0.125*(W+W(-,))

V        = TWO_DP_DT*SUM(QBAR*AJ_D)
```

in which AJ_D contains $\epsilon J_{i,j-\frac{1}{2}}$. (This procedure is probably unnecessarily compli-

cated. A simpler, and faster, approximation to the current would probably be sufficient.)

The final step in each iteration is to up-date the value of χ at the coordinate axis, using eqn. 13. In DAP Fortran, with obvious notation

```
CHI_AXIS = (SUM2_1D+SUM3_1D+SUM2D)/SUM1_1D

CHI(1,)  = VEC(CHI_AXIS)
```

In the code, the iterations are performed in cycles of 50 iterations. After each cycle, the maximum change in χ ($\Delta\chi$) and the maximum residual are computed. The equilibrium equation is considered solved when $\Delta\chi$ is less than some specified tolerance.

4.8 Location of the magnetic axis

The magnetic axis $R = R_m$ is the point at which χ attains its minimum value. Under the hypothesis that χ is monotonic and has up/down symmetry, the (unique) magnetic axis lies along $Z = 0$. We locate it using quadratic interpolation, as follows.

Let R_1 be the position along $Z = 0$ of the minimum mesh value of χ and let R_0 and R_2 be the positions of the mesh points on either side (along $Z = 0$) such that $R_0 < R_1 < R_2$.

The Lagrangian form of the quadratic interpolating polynomial $\chi(R)$ may be written

$$\chi(R) = \frac{(R_2 - R_1)(R - R_1)(R - R_2)\,\chi(R_0)}{D} - \frac{(R_2 - R_0)(R - R_0)(R - R_2)\,\chi(R_1)}{D}$$

$$+ \frac{(R_1 - R_0)(R - R_0)(R - R_1)\,\chi(R_2)}{D}$$

$$D \equiv (R_1 - R_0)(R_2 - R_1)(R_2 - R_0).$$

Differentiating this and setting the result to zero, we find that the minimum of $\chi(R)$ occurs where

$$R = \tfrac{1}{2} \cdot \frac{\chi(R_0)(R_2 - R_1)(R_2 + R_1) + \chi(R_1)(R_0 - R_2)(R_0 + R_2) + \chi(R_2)(R_1 - R_0)(R_1 + R_0)}{\chi(R_0)(R_2 - R_1) + \chi(R_1)(R_0 - R_2) + \chi(R_2)(R_1 - R_0)} \qquad (18)$$

This is therefore the position of the magnetic axis.

In the mapping onto the DAP (Fig. 2), the line $Z = 0$ occupies two non-adjacent columns (2 and 63). The first step in evaluating eqn. 18 is to extract the values of χ and R along yhe line $Z = 0$, ordering the increasing R when regarded as a long vector. Account has to be taken of the fact that the axis value is the first element in both columns. For example, in order to extract the values of χ, we write

```
WORK_CHI(,) = 1.0E10
V()         = CHI(,63)
V()         = V(+)
V(64)       = 1.0E10
WORK_CHI(,1)=REV(V)
WORK_CHI(,2)=CHI(,2)
```

The position of the minimum mesh value of χ is located using the built-in function MINP:

```
L(,) = MINP(WORK_CHI).
```

The values of χ_0, χ_1, χ_2 and R_0, R_1, R_2 are now extracted. For example χ_1 and R_1 are obtained from

```
R_VALS(2)    = WORK_R(L)

CHI_VALS(2)  = WORK_CHI(L)
```

and similarly for the values of χ_0, R_0 and χ_2, R_2 (using left and right shifts in long vector ordering). The vectors R_VALS and CHI_VALS then contain the values

R_VALS: $R_0, R_1, R_2, R_0, \ldots\ldots\ldots$

CHI_VALS: $\chi_0, \chi_1, \chi_2, 0, \ldots\ldots\ldots$

The sums and differences of the Rs are now computed in parallel and re-ordered:

```
V() = R_VALS() + R_VALS(-)

R_SUMS() = 0.0E0

R_SUMS(ELS(1,2)) = SHLP(V,2)

R_SUMS(3) = V(2)
```

and similarly for R_DIFFS. The vectors R_SUMS and R_DIFFS now contain the values

R_SUMS: $R_2 + R_1, R_0 + R_2, R_1 + R_0, 0 \ldots\ldots\ldots$

R_DIFFS: $R_2 - R_1, R_0 - R_2, R_1 - R_0, 0 \ldots\ldots\ldots$

The components of the top and bottom of eqn. 18 are computed from

```
BOTTOM( ) = CHI_ VALS( )*R_DIFFS( )

TOP       = BOTTOM( )*R_SUMS( )
```

and the position of the magnetic axis obtained from

```
R_MAG_AXIS = 0.5E0*SUM(TOP)/SUM(BOTTOM)
```

4.9 Determination of the accuracy of the ψ surfaces

The ψ surfaces should eventually lie in the χ surfaces. To measure the accuracy of a given mesh, we first compute the average value of χ around each ψ surface,

$$\bar{\chi}_j = \frac{1}{64} \sum_{i=1}^{64} \chi_{i,j}$$

and then define the surfaces error e_s by

$$e_s = \max_{i,j} |\chi_{i,j} - \bar{\chi}_j|$$

The equilibrium problem is considered solved when e_s is less than some specified tolerance.

In DAP Fortran,

```
CHI_AV(,) = MATC(SUMC(CHI)/64,0E0)

SURFACE_ERR = MAXV(ABS(CHI–CHI_AV)).
```

4.10 Respacing the ψ mesh

This is the most interesting aspect of the implementation on the DAP since at first sight it might appear to be extremely difficult to do in parallel. However, as shown below, application of the useful rule 'organise the data, then do the arithmetic' makes this operation quite simple.

Respacing the ψ mesh may be described as follows. As a result of solving the equilibrium equation on some mesh, values of $R_{i,j}$, $Z_{i,j}$, $\chi_{i,j}$, are known. A set of new χ values $(\chi_{new})_k$ $2 \leq k \leq 63$ is then chosen. These are to be the $\chi(\psi_k)$ on the adjusted mesh. The values of $R_{i,k}$ and $Z_{i,k}$ corresponding to the $(\chi_{new})_k$ are then found using quadratic interpolation (along each θ ray) based on the known values $R_{i,j}$, $Z_{i,j}$, $\chi_{i,j}$. In terms of Fig. 2, this can be thought of as respacing the $R_{i,j}$, $Z_{i,j}$ points down each column i.

To see what data is required, we next discuss the interpolation. This is conveniently done using Neville's algorithm; given a function $f(\chi)$ at points χ_0, χ_1, χ_2 ($\chi_0 < \chi_1 < \chi_2$), the linear interpolating polynomials $p_{0,1}(\chi)$ and $p_{1,2}(\chi)$ are first found from

$$p_{0,1}(\chi) = \frac{(\chi-\chi_0)f(\chi_1) - (\chi-\chi_1)f(\chi_0)}{(\chi_1-\chi_0)}$$

$$p_{1,2}(\chi) = \frac{(\chi-\chi_1)f(\chi_2) - (\chi-\chi_2)f(\chi_1)}{(\chi_2-\chi_1)}$$

and the quadratic interpolating polynomial is computed from

$$p_{0,2}(\chi) = \frac{(\chi-\chi_0)p_{1,2}(\chi) - (\chi-\chi_2)p_{0,1}(\chi)}{(\chi_2-\chi_0)} \tag{19}$$

(In our application, p represents R and Z).

For each $(\chi_{new})_k$ and each θ_i, we therefore need to extract the appropriate values of $(R_{i,0}, R_{i,1}, R_{i,2})$, $(Z_{i,0}, Z_{i,1}, Z_{i,2})$ and $(\chi_{i,0}, \chi_{i,1}, \chi_{i,2})$. Inspection of the error term of the interpolating polynomial shows that greatest accuracy is obtained when $\chi_{i,1}$ is the value nearest $(\chi_{new})_k$. To locate this value of $\chi_{i,1}$, we find the smallest value of j (say l) for which the inequality

$$(\chi_{new})_k < \tfrac{1}{2}(\chi_{i,j} + \chi_{i,j+1}) \qquad \begin{matrix} 2 \le j \le 63 \\ 2 \le k \le 63 \end{matrix} \tag{20}$$

is satisfied. Note that $l = l(i,k)$ since (apart from its obvious dependence on k) the inequality 20 will be satisfied at different positions along different θ rays. The values $\chi_{i,l(i,k)-1}$, $\chi_{i,l(i,k)}$, $\chi_{i,l(i,k)+1}$ are then used as the values $\chi_{i,0}$, $\chi_{i,1}$, $\chi_{i,2}$ for the interpolation (and similarly for R and Z).

To perform this operation on the DAP, we first compute the array of midpoint values used in eqn. 20, and give it dummy values along rows 1 and 63, so that interpolation centered on rows 1 and 64 is not attempted:

```
CHI_MID(,) = 0.5E08(CHI+CHI(+,))

CHI_MID(1,) = -1.0E10

CHI_MID(63,) = 1.0E10
```

The values of χ_{new} are then extracted,

```
V_NEW_CHI( ) = CHI(,2)
```

For each $(\chi_{new})_k$, inequality 20 is evaluated, using a logical matrix to mark the points where it is satisfied.

```
DO 10 K = 2,63

L(,) = MAT(V_NEW_CHI(K)).LT.CHI_MID(,)
```

The first .TRUE. point down each column is the point $l(i,k)$ referred to above. This is extracted from the logical matrix **L** using the built-in DAP Fortran row-number function, ROWN:

```
IV( ) = ROWN(L)
```

in which **IV** is an integer vector to hold the $l(i,k)$ for each column i. The values $\chi_{i,l(i,k)}$ etc. are then extracted using the row-number addressing construct. For example, for $\chi_{i,1}$, we have

```
CHI1(K,) = CHI(IV,)
```

When this process has been repeated for all k, the $\chi_{i,j}$ values are updated

```
CHI(,) = MATC(V_NEW_CHI)
```

Finally, the arithmetic in eqn. 19 is done in parallel. For example, in obvious notation

```
C_M_C0 = CHI–CHI0
  ...       ...

P_01 = (C_M_C0*R1_C_M_C1*R0)/C1_M_C0
P_12 = (C_M_C1 *R2_C_M_C2*R1)/C2_M_C1
R(MASK) = (C_M_C0*P_12_C_M_C2*P_01)/C2_M_C0
```

and similarly for Z. This completes the respacing of the ψ mesh.

5 Results

Figure 3 shows the χ surfaces in JET geometry obtained from a run on the QMC DAP. For clarity, the theta lines have been omitted. The input parameters were:

boundary shape:	R_0	= 2·96
	a	= 1·25
	δ	= 0·2539
	ϵ	= 1·68
p and F:	p_0	= 10^6 N/m^2
	α	= 2
	β	= 2
current and field:	I	= 3·8 MA
	B_0	= 2·77 T

Numerical: $\omega = 1.9$
$\Delta\chi = 4\ 10^{-5}$
$e_s = 10^{-3}$

The values of ω, $\Delta\chi$ and e_s were arbitrary first choices. We have not attempted to study the effects of varying them.

The code was run using 32 bit precision. The relevant numerical results for this case are given below. [Note that the location of the magnetic axis was repeated three times ('axis iterations') as a diagnostic check on the accuracy. This extra computation is strictly unnecessary.]

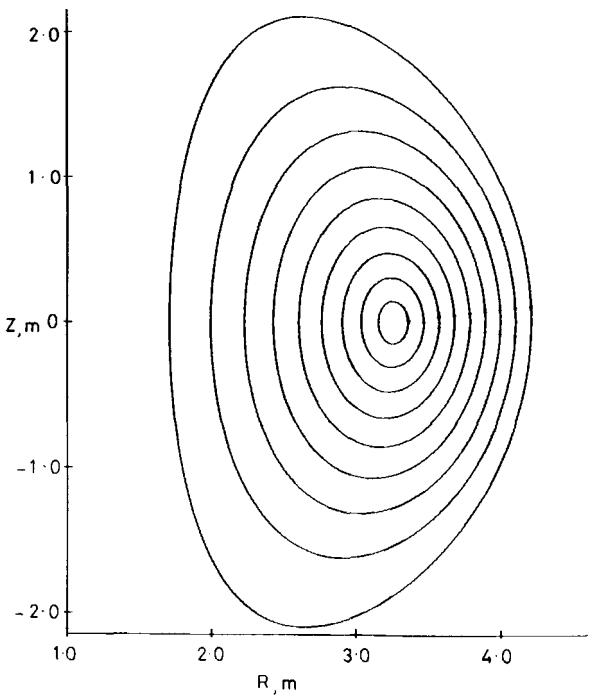


Fig. 3 A JET equilibrium computed on the ICL DAP

Values obtained when locating the magnetic axis:

Axis iteration	SOR cycles	Maximum change	Maximum residual	Magnetic axis	Gamma
1	39	$2{\cdot}0027\ 10^{-5}$	$3{\cdot}3734\ 10^{-6}$	3·2512	$2{\cdot}0986\ 10^{-2}$
2	32	$3{\cdot}2425\ 10^{-5}$	$3{\cdot}7793\ 10^{-6}$	3·2502	$2{\cdot}1215\ 10^{-2}$
3	4	$1{\cdot}9073\ 10^{-5}$	$5{\cdot}6216\ 10^{-6}$	3·2507	$2{\cdot}1219\ 10^{-2}$

Values obtained for the surface iterations:

Metric iteration	SOR cycles	Maximum change	Maximum residual	Surface error	Gamma
1	2	$3{\cdot}9101\ 10^{-5}$	$3{\cdot}4798\ 10^{-6}$	$9{\cdot}8725\ 10^{-2}$	$2{\cdot}1218\ 10^{-2}$
2	15	$3{\cdot}0518\ 10^{-5}$	$4{\cdot}1756\ 10^{-6}$	$1{\cdot}9211\ 10^{-4}$	$2{\cdot}1190\ 10^{-2}$

This calculation took approximately 180s of DAP processing time. Since we have not written a conventional Fortran equivalent of the DAP code, an exact timing comparison with other machines cannot be made. However, for orientation, a similar calculation using the code described in Delucia *et al.*[3] took approximately 5200s (a factor of 29) on an ICL 2976.

6 Summary

We have demonstrated the solution of a substantial computational problem on the production ICL DAP. No difficulties were encountered with the DAP hardware or software. The computational operations needed for the calculation were easily programmed in DAP Fortran. The DAP code ran approximately 29 times faster than a similar (but not identical) conventional Fortran code on an ICL 2976.

Acknowledgments

I should like to thank the Computer Centre of Queen Mary College for the use of the DAP. In particular, I should like to thank G.Bowgen of the DAP Support Unit for his assistance in running the code.

Appendix

Derivation of the axisymmetric plasma equilibrium equation in ψ, θ coordinates. In arbitrary coordinates x^i, **B** = curl **A** is

$$B^i = \frac{1}{\sqrt{g}} \epsilon^{ijk} \frac{\partial A_k}{\partial x^j} \tag{A1}$$

in which g is the determinant of the metric tensor. Let x^3 be the ignorable co-ordinate corresponding to axisymmetry. Then the components of (A1) are

$$B^1 = \frac{1}{\sqrt{g}} \frac{\partial A_3}{\partial x^2}$$

$$B^2 = -\frac{1}{\sqrt{g}} \frac{\partial A_3}{\partial x^1} \tag{A2}$$

$$B^3 = \frac{1}{\sqrt{g}} \left(\frac{\partial A_2}{\partial x^1} - \frac{\partial A_1}{\partial x^2} \right)$$

Using eqn. A2, the equations for the field lines,

$$\frac{dx^1}{B^1} = \frac{dx^2}{B^2} - \frac{dx^3}{B^3}$$

give

$$\frac{\partial A_3}{\partial x^1}\,dx^1 + \frac{\partial A_3}{\partial x^2}\,dx^2 = 0$$

or

$$A_3 = \text{constant.}$$

The magnetic surfaces are therefore defined by A_3 = const. Similarly, the current surfaces are defined by B_3 = constant. For convenience, we define

$$\begin{aligned} \chi &\equiv A_3 \\ F &= B_3 . \end{aligned} \tag{A3}$$

The expressions for the magnetic field and the current may then be written

$$B^1 = \frac{1}{\sqrt{g}}\frac{\partial \chi}{\partial x^2}$$

$$B^2 = -\frac{1}{\sqrt{g}}\frac{\partial \chi}{\partial x^1}$$

$$\mu_0 j^1 = \frac{1}{\sqrt{g}}\frac{\partial F}{\partial x^2} \tag{A4}$$

$$\mu_0 j^2 = -\frac{1}{\sqrt{g}}\frac{\partial F}{\partial x^1}$$

$$\mu_0 j^3 = \frac{1}{\sqrt{g}}\left(\frac{\partial B_2}{\partial x^1} - \frac{\partial B_1}{\partial x^2}\right)$$

The components of the equilibrium equation j x B $=\nabla$ p are

$$\sqrt{g}\,(j^2 B^3 - j^3 B^2) = \partial p/\partial x^1 \tag{A5}$$

$$\sqrt{g}\,(j^3 B^1 - j^1 B^3) = \partial p/\partial x^2 \tag{A6}$$

$$\sqrt{g}\,(j^1 B^2 - j^2 B^1) = 0 \tag{A7}$$

Using eqn. A4 in eqn. A7, we obtain

$$\frac{\partial F}{\partial x^2}\frac{\partial \chi}{\partial x^1} - \frac{\partial F}{\partial x^1}\frac{\partial \chi}{\partial x^2} = 0$$

so that $F = F(\chi)$. Multiplying eqn. A5 by B^2, eqn. A6 by B^1 and adding, we obtain

$$\frac{\partial p}{\partial x^2}\frac{\partial \chi}{\partial x^1} - \frac{\partial p}{\partial x^1}\frac{\partial \chi}{\partial x^2} = 0$$

so that $p = p(\chi)$. Eqn. A5 may then be rewritten to give

$$\frac{\partial B_1}{\partial x^2} - \frac{\partial B_2}{\partial x^1} = -\sqrt{g}(F_\chi B^3 + \mu_0 p_\chi) \qquad \text{(A8)}$$

This is the equilibrium equation in tensor form.

In magnetic surface coordinates ψ, θ, ϕ, ψ and θ are both orthogonal to the toroidal angle ϕ, but not to each other. The metric tensor therefore has the form

$$\begin{pmatrix} g_{11} & g_{12} & 0 \\ g_{21} & g_{22} & 0 \\ 0 & 0 & g_{33} \end{pmatrix}$$

in which $g_{33} = R^2$. It follows that $B^3 = F/R^2$. The remaining g_{ik} can be calculated as derivatives of R and Z by a covariant transformation from cylindrical coordinates:

$$g_{ik} = g_{\mu\nu}\frac{\partial x^\mu}{\partial x'^i}\frac{\partial x^\nu}{\partial x'^k}$$

$$x'^i = \{\psi, \theta, \phi\} \quad x^\mu = \{R, Z, \phi\}$$

$$g_{\mu\nu} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & R^2 \end{pmatrix}$$

The result is

$$g_{11} = R_\psi{}^2 + Z_\psi{}^2$$
$$g_{22} = R_\theta{}^2 + Z_\theta{}^2$$

$$g_{12} = R_\psi R_\theta + Z_\psi Z_\theta$$

$$\sqrt{g} = R(R_\psi Z_\theta - R_\theta Z_\psi) \equiv J.$$

Then, defining

$$h_{\psi\psi} = g_{22}/\sqrt{g} = (R_\theta{}^2 + Z_\theta{}^2)/J$$

$$h_{\theta\theta} = g_{11}/\sqrt{g} = (R_\psi{}^2 + Z_\psi{}^2)/J$$

$$h_{\theta\psi} = -g_{12}/\sqrt{g} = -(R_\psi R_\theta + Z_\psi Z_\theta)/J,$$

Eqn. A8 may be written in its final form

$$\frac{\partial}{\partial\psi}(h_{\psi\psi}\chi_\psi + h_{\theta\psi}\chi_\theta) + \frac{\partial}{\partial\theta}(h_{\theta\psi}\chi_\psi + h_{\theta\theta}\chi_\theta) = -J\left(\frac{FF_\chi}{R^2} + \mu_0 p_\chi\right).$$

References

1. CALLEN, J.D. and DORY, R.A.: *Phys. Fluids* 1972, **15**, 1523
2. LACKNER, K: *Comput. Phys.* 1976, **12**, 33.
3. DELUCIA, J., JARDIN, S.C. and TODD, A.M.M.: 'An iterative method for solving the inverse tokamak equilibrium problem', Princeton Plasma Physics Laboratory Report PPPL 1564, July 1979.