SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD & APPLETON LABORATORIES

COMPUTING DIVISION

D I S T R I B U T E D C O M P U T I N G N O T E 4 6 2

Visit : ICL RADC, Stevenage
- 5 August 1981

issued by
R W Witty
K Jeffery

13 August 1981

DISTRIBUTION:

F R A Hopgood
G Manning
D A Duce
R W Witty
K Jeffery

1. INTRODUCTION

Bob Hopgood, Keith Jeffery, Rob Witty and Roger Vinnicombe (ICL Marketing) visited ICL's Research and Advanced Development Centre (RADC) at Stevenage with a view to strengthening the collaborative link between ICL and RAL. Gordon Scarrott retired on 8 May 1981, and since that day it has been run by Vic Maller. Mr Maller gave a presentation of the new organisation of ICL's research and development activities and included details of RADC activities.

2. CURRENT ORGANISATION

With the disbanding of Product Development Group (PDG), West Gorton has become responsible for big systems and Bracknell has become responsible for small distributed systems. Both report directly to Wilmot. Bill Talbot's Division is now called Research and Technology Group and is also responsible for Product Planning. An outline of the new structure is given in Figure 1.

Charlie Portman has been moved sideways into Software Technology Division. This essentially runs Design Automation. Norman Brown takes over from Charlie Portman but in addition takes over responsibility for RADC, Stevenage, as well as the West Gorton Advanced Development Centre, which is now run by Chris Burton.

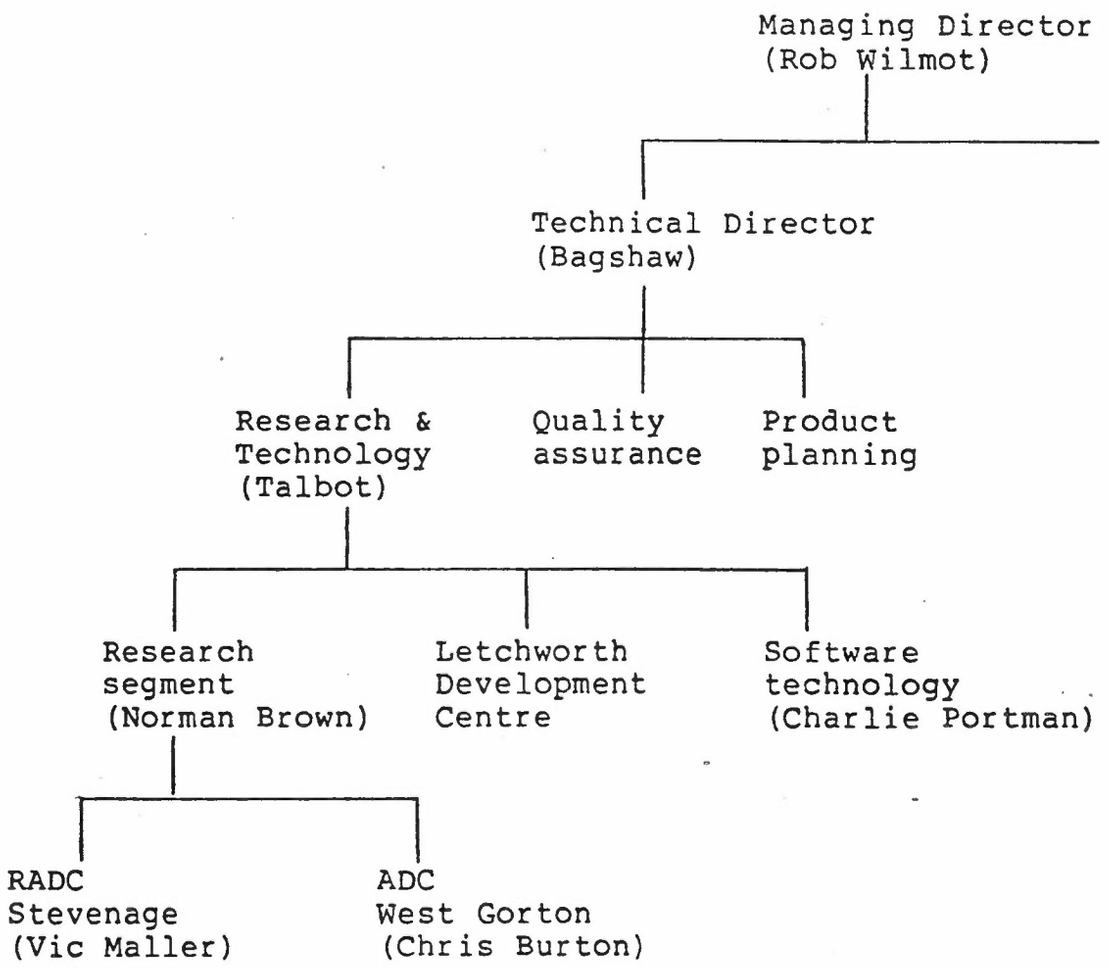


Figure 1

ICL R&D Organisation

3. RADC BACKGROUND

RADC was originally an ICT institution created in the early 60s and concentrating on hardware technology. In 1967 Gordon Scarrott took over and merged RADC with the Ferranti Research and Development activity. From 67 onwards, RADC was primarily concerned with the John Illiffe work. In 1970 the English Electric R&D activity was merged with the Stevenage activity and this brought Mike Underwood and Stuart Reddaway into RADC. At this time, two basic activities were the basic language machine from Illiffe and pattern recogniton work from English Electric. In 1971, they began the CAFS project with ACTP money, and the current format is that they have a wide ranging brief to do research and development activity to a large extent independent of product development, having a modest budget. To do any larger projects, they must bid to the Company or to DoI for separate funding. They currently have four major lines of research, all of which are supposedly non von Neumann architectures. There is very little basic software research done at RADC. The overall assumption is that silicon technology will eventually mean that processors become specialist tools, designed to do special applications, and the very general von Neumann architecture will fade away. With this background, their four projects are:

1. SIMD.
   This is the DAP (Distributed Array Processor) project of Reddaway.

2. Man-Machine Interface.
   This is the pattern recognition and speech input/output work of Underwood.

3. Storage and Data Management.
   This is the CAFS work of Vic Maller.

4. Functionally Distributed Architectures.
   This is essentially the multi-processor shared memory system of Owen Evans which has grown out of the basic language machine work of Illiffe.

4. MAN-MACHINE SYSTEMS

RADC are currently working on three projects to try and improve the man-machine interface.

1. Hand-held terminal

2. Speech interaction

3. Knowledge engineering.

5. SPEECH INTERACTION.

Mike Underwood introduced us to the speech work at RADC. The objectives of this work are to produce a speech system which is

(i) low cost
(ii) real time, ie, no undesirable delays
(iii) easy to use by the end user
(iv) easy to use by the programmer
(v) telephone bandwidth, so that it has a very wide application

5.1 Speech Recognition

Mike Martin demonstrated the speech recognition system. It uses a telephone handset carbon microphone for input. Carbon microphones are not the best available, but the whole system is geared to work over the ordinary Post Office telephone network. The current system did not take account of telephone level noise. It was trained on several individuals rather than keyed to one individual, and it does single word recognition by pattern recognition techniques. It tried to recognise the beginning and end of the word, and then goes through pattern recognition technique to clarify the entire word. Using the pattern recognition approach, means that no assumptions are made about the distribution or frequency of occurrence of the words. This gives better performance, but requires more training. Current vocabularies were around 64 words.

RADC had built three recognition systems, the Mark I, followed by a 16-channel device, followed by a current Z-80 based system. The current recogniser uses a 2 MHz Z-80 processor. The system was demonstrated to us and it was clear that whilst the system was quite good at recognising even difficult vocabularies like the letters of the alphabet, it was by no means in the 95% correct region. RADC are very aware of the human factors in the systems level aspects of all speech work, and it was very clear that these aspects are very very important indeed. For example, the recogniser will fail to recognise someone who is speaking whilst under emotional strain, because it alters the quality of the voice. So under no circumstances does one have the stop command, or the "bombs away" command under voice recognition, if these are likely to be used in stressful situations! Our general impression of this work was that it was as good as anything around, and we were quite impressed by the RADC's approach.

5.2 Speech output

Speech output is naturally slightly easier than speech input. For example, speech synthesis runs at 3,000 bits per second, whereas speech input goes at 30 KHz. There are four approaches to generating speech output.

(i) Output pre-recorded in wave forms.
(ii) Formant synthesis
(iii) Synthesis by rule
(iv) Synthesis from text

The use of pre-recorded waveforms gives good quality output of individual words, phrases, but can lead to problems because the rythymn and intonation of the whole conversation is likely to break the normal human rules of speech, ie, misunderstandings can occur because of the strange inflections, etc. Synthesis by rule is based on a linguistic premise that a spoken utterance can be described in terms of discrete sounds in the same way the message can be constructed from the letters of the alphabet. This is obviously a very powerful technique, but currently it is too expensive and sometimes cannot run in real time, so it is still very much a blue skies research idea; as is synthesis from text, which is an even more general problem. Therefore RADC have tended to go for Formant synthesis.

Pre-recorded waveforms demand an output rate of around 20,000 bits per second. Formant synthesis reduces this data rate significantly by describing the required speech as a model of the acoustic behaviour of the vocal tract in terms that are related to what the speaker does with his vocal apparatus when speaking, rather than to capture the detailed waveform of the spoken sound. What happens when the air flows from the lungs through the vocal cords and through the articulators, which are the tongue, the lips and the jaw, can be captured as a set of parameters. The storage of these parameters is much less than storing the actual waveform of the speech itself, because the key factor in format synthesis is the generation of the vocabulary. ICL have developed some useful tools to enable pre-recorded sound to be then modified and edited by interactive programs to produce good parameters. The ICL system uses ten parameters which must be changed at the rate of 100 times per second to produce reasonable speech.

ICL are aiming at developing a system whereby the building of new vocabularies can be done by non-experts. ICL demonstrated two output devices. The more modern was called ORAC, which is a Z-80 based system. It is orientated towards the telephone system and we were given a demonstration of the Post Office short code dialling service. This is currently on test in the Post Office R&D Department. The motivation behind producing a microprocessor-driven system was to produce one which can accept a logical level of interaction dialogue - that is, it should be able to handle chunks of dialogue with the user, so that the mainframe delays are isolated. The ORAC system can handle voice input or digital input via a touch toe keypad. The ORAC controller performs three types of function :

(i) detailed control of all the I/O, maintaining data flow to the synthesiser, receiving data from the touch toe detector, controlling the line unit and servicing the communications protocol.

(ii) providing the high level interface for control of speech output.

(iii) providing the interaction management, independent of the main frame.

The Formant synthesiser approach was the style of output to be controlled. It is envisaged that different levels of control are appropriate to different applications, so a number of levels have been implemented. At the lowest level each vocabulary item can be called by number. These items can be of any length and any one can be a phrase a word, or part of a word. This is the most flexible arrangement, and enables the best compromise to be made between naturalness and the efficient use of the store. If all the words in a particular phrase appeared nowhere else

appeared nowhere else in a vocabulary, the most natural utterance results from storing it as a complete phrase. If a word, or part word occurs in several contexts, more efficient use can be made of memory by storing the item once only, and concatenating it with different vocabulary items to form complete words. It is likely that when a vocabulary item is used in different contexts, it needs to be emphasised differently. For example, the "six" in the word "six" is shorter than the "six" in the word "sixteen".

To accommodate changes in emphasis, optional control characters can be used to control the pitch and speed of each item independently. Changes in the rate of speaking of the synthesiser cannot be applied uniformly to the speech, so that the vocabulary data contains information that allow speed changes to be carried out in the appropriate manner. Whilst this level of control provides the programmer with a powerful way of changing the speech and at the same time shelters him from the details of how it is done, there is a need for some commonly used strings of vocabulary items to be handled more easily. One example is the different forms of numerical output which are likely to form part of most application vocabularies. To save the programmer from having to remember to use explicitly the rules for generating quantities, the machine should say "thirty-three" not "Three-ty Three". These words are coded in the controller and enable the different types of output to be generated automatically, thus n(digit string) causes the digits to be output with suitable pauses and prosodic changes to make them easily understood, eg 1933 O(digit string) gives the ordinal form, eg, "one thousand nine hundred and thirty-third" q(digit string) causes the number to be spoken as a quantity, eg, "one thousand nine-hundred and thirty-three". d(digit string) causes the number to be spoken as a date, eg "nineteen thirty-three".

A particularly important requirement for a speaking system is that the listener must be able to cause the last message to be repeated. It is a natural human reaction when asked to repeat something for the speaker to say it again more slowly and deliberately, so that the listener will have a better chance of understanding it the second time. The controller is provided with a facility which mimics this aspect of human behaviour.

RADC were aiming their work at applications which use the ordinary public telephone, such as

1. Order entry by mobile salesman
2. Status reporting, for example work in progress
3. Banking enquiries and transactions
4. Credit authorisation
5. Spares location
6. Ticket enquiry and reservations
7. Direct mail order

Most of these systems use remote data entry or database enquiry, where the voice is the sole means of communication with the user. So far speech output has not been integrated into existing terminal facilities (this is just becoming into use with things like the PERQ). There are many instances where the use of voice could provide a valuable adjunct to ordinary terminal facilities. Some initial applications might be to

(i) provide verbal feedback of data that has just been entered, allowing complementary use of eyes and ears in checking;

(ii) provide instructions to operators in a broadcast manner where direct line of sight is not possible;

6. DISTRIBUTED SYSTEMS

Owen Evans introduced this work and felt that ICL's idea of distributed systems was tightly coupled MIMD systems where many processors were tightly coupled together to act as a single computing system. Their work had its origins in a pipeline of sequential processes running on several processors, using shared store to reduce message traffic. They have produced a design for up to 16 processors connected together by shared store but with their own local memory, very much the same as the Cover M design CYB-M. Currently the machine uses 8-bit microprocessors and has 8 processors installed, giving a potential speed of around 8 Mibs with 8 processors. The shared store runs at 1 megabyte per second and is connected to the processors by an inter-processor bus. The work assumes a low inter-process communication rate. Access time for the shared memory is twice that for the local memory. The raw memory accesses go through an indirection mechanism which gives some kind of protection. They assume that 1 in 20 store accesses will be to the shared memory. Multi-programming is possible within one processor. Messages between processes can be queued and any queue can serve more than one process. They currently have static and manual process allocation. Each processor has 64 Kbytes of local store and the shared store can be up to 1 Megabyte in size. Memory mapping is achieved by the indirect memory access mechanism. Each processor has a kernel operating system in it, the whole thing being incredibly similar to CYBA-M except that it does not appear to have some of the nice debugging facilities of CYBA-M.

7. CAFS

See Appendix 1

8. DAP

Stewart Reddaway gave us a quick update on the current state of his DAP work. They are doing quite a lot on applications of the DAP. They have a study contract for example from RSRE Malvern to look at replacing lots of special purpose FFT-type processor boxes, with one DAP to do everything. This is called the multi-mode airborne radar study contract. They have been looking at using the DAP to design automation from VLSI design rule checking and the tracking problem. They have been experimenting with the Queen's problem with parallel table look-up, with parallel compiling at the University of Kent, and with sorting and file-searching with the object of trying to sell a DAP to the clearing Banks. Reddaway himself is working on the super DAP.

8.1 Super DAP

The Super DAP is a paper study which is aimed at increasing the floating point power of the DAP. The ordinary DAP is weak on floating point computnation, taking over a thousand cycles per floating point multiply. In the Super DAP, each processing element has now grown a 128-bit register file which can be locally addressed. Store cycles are separate from this local register file access, so that the PE can run much faster. With this arrangement, the Super DAP can do a floating point multiply 320 times faster than the ordinary DAP. It is 640 times faster for a floating point add.

With the advent of the ANSI work on unconstrained array Fortran, the DAP people are now talking in terms of linear numbers of processors, rather than two-dimensional numbers of processors.

8.2 Nibble mode

The INMOS 64K ram chip (IMS 600) has an extra feature on it called nibble mode. It takes 150 nanoseconds to get the first bit out of its memory but the next three bits can be delivered at one every 50 nanoseconds. The Super DAP is possibly being designed to take account of this nibble mode in order to work on 4 bits at a time, rather than one bit at a time. They have also been looking at using this chip to enable a 16 x 16 DAP to be put onto two boards using VLSI technology. It would seem that the Super DAP loses a lot of the essential elegance of the one-bit DAP, and is rather a technological hack in order to cure the floating point problem and exploit the new memory chip.

9. KNOWLEDGE ENGINEERING

Tom Addis introduced RADC's work on Knowledge Engineering. RADC have been experimenting with Knowledge Engineering to produce an Expert System which would act as a faults database and diagnosis expert. They have produced two systems - one called CRIB, and one called RAFFLES. Tom Addis felt that an expert when solving a problem went through two phases.

Firstly, he asked a fixed sequence of questions, and then if the fixed sequence of questions failed to produce an answer, the expert went into more general problem solving mode. Both of these two modes relied on some kind of pattern matching mechanism to help recognise when a solution had been found, or to generate advice as to what to do next. CRIB is a pattern matching system. In the fault diagnosis example, the engineer feeds in the symptoms of the problem to CRIB, which will reply with an ordered list of tests for the engineer to perform to try and home in on the exact fault, and the symptoms are matched against patterns of known faults. To try and generate this strategy, the system is able to learn if the particular tests applied have solved the problem, ie, it is adaptive.

The RAFFLES system helps the engineer go through the fixed sequence of questions. It generates a hierarchic structured list of questions and leads the engineer through this hierarchy to try and home in on the fault. Tom Addis only had a short amount of time to present this work, and it was not really clear whether the systems really were expert and really could learn. Tom Addis himself is leaving RADC soon to join Brunel University. However, he will be retaining his links and doing a joint project between RADC, himself at Brunel, and Imad Thorsun at Imperial College.

10. ACTIONS

1. Rob Witty to get David Duce to get Aspinall's CYBA-M people to talk to the RADC people about shared memory systems.

2. Rob Witty to link Tom Addis into the SERC's Software Engineering initiative. Addis is going to work on program synthesis at Brunel.

3. Bob Hopgood/Rob Witty to see if we can get hold of the QMC speech kit and to further explore the possibility of RADC collaborating with us on PERQ speech.

4. ICL to send us further papers on their work.

INTERNATIONAL COMPUTERS LIMITED
RESEARCH AND ADVANCED DEVELOPMENT CENTRE

VISIT BY PROFESSOR F.R.A. HOPGOOD
ON WEDNESDAY, 5TH AUGUST 1981

A G E N D A

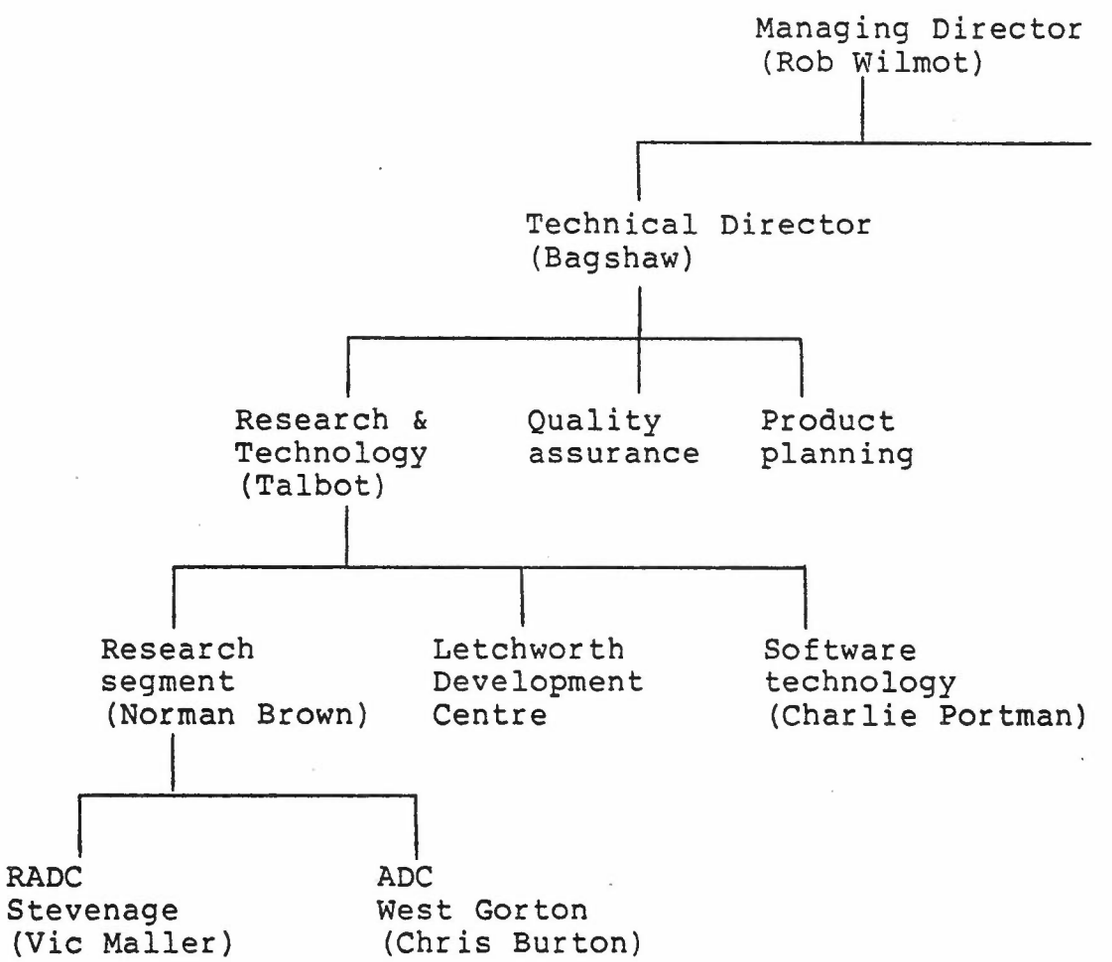
10.00	Arrive, Coffee, General Introduction	V.A.J. Maller
10.30	Speech	M.J. Underwood
12.30	Lunch	
13.30	Distributed Systems	O.V.D. Evans
14.15	CAFS Update	V.A.J. Maller
15.00	DAP Update	S.F. Reddaway
15.45	Knowledge Engineering	T.R. Addis
16.30	Final Discussion	All
	Depart	

MJU/ed
4.8.81.

Notes on visit to ICL Research and Advanced Development Centre
Wednesday 5 August 1981

Introduction

Bob Hopgood, Rob Witty and Keith Jeffery visited ICL RADC at Stevenage. RADC is led by Vic Maller, and fits into the hierarchy:-



The terms of reference of RADC are to indulge in research and development relevant to computing. The research interests are machine architectures, data management and man-machine interaction represented by DAP and distributed systems, CAFS and work on speech respectively. The whole area of work is non-Von-Neumann processing, and they see hardware and software as being non-independent. All their activities use specialised hardware and software.

Speech I/O: Mike Underwood, Mike Martin

Speech applications include

- users attention occupied	eg operator
- users hands occupied	eg data entry
- remote terminals	via telephone - every telephone a terminal!

- unskilled users — eg factory managers
- handicapped people — eg talking calculator facility
- no other way — eg 'speak and spell'

Objectives

- low cost
- real time
- easy to use by programmer and end user
- telephone integral part of system

The hardware for speech input is a telephone handset, and a speech recognition unit, together with a VDU to output the results. The system is trained to learn up to 64 words. The method of recognition is to recognise the start and end of the word. The frequency (200-900Hz) is resolved into 16 bins. The energy is resolved into 3 bands. The word is then divided into 8 equal slots; each slot has its data reduced to statistical form (median, mode) and this gives the measurement vector for the word. This reduces to 16 4-bit numbers, 8 amplitudes, 8 frequencies, plus 1 4-bit number representing length of the word. The system is robust, and seems capable of recognising words about 80% of the time.

For speech output they are working on a synthesizer which is driven by a description of the voice. The input is parameters which describe the sound - now get this generated by the synthesizer. The output phrase is controlled by parameters - affecting pitch and frequency - which allow control of intonation by affecting segments of the phrase. The system is most impressive and is very effective. There are many obvious applications where instructions to a user of a computer terminal are required.

Distributed Systems: Owen Evans, Mike W Martin

Working on distributed processing in a single box - a group of cooperating processors working on one or more cooperative processes.

Intercommunication is through a shared store area.

Engine of 16 processors, each has its own local store and shared store. (Note: not unlike the Aspinall machine at UMIST, or the dataflow machine at Manchester; both DCS projects).

The idea is to separate problems into processes, and to distribute the processes among the processors by function. The hardware is 16 8 bit processors, each with 64Kb store and there is 1Mb of shared store. The level of indirection means that there is no real limit. The kernel software acts as one master process in all processors.

There is no description of the interconnection of processes and so resource sharing is difficult. The topology definition is at compile-time, so there is a definite lack of flexibility.

CAFS: Vic Maller

CAFS 800 runs on DME 2946, 50, 55, 60. Hardware is Channel controller plus modified EDS60 drives. There is a parallel head read of 10 surfaces, but the drive is switched to single head read for writing. The channel logic works at 4-5 Mb/sec on read, but the discs can only pump out 300 Kb/sec per head.

Now moving to a development with no parallel head activity because new drives are fast enough and because this overcomes the problem of single head write/multi head read and also the restriction of records to track length. In brief, the intelligence will move from the channel to the drive so allowing standard hardware.

CAFS gives high speed filtering in the data stream. It is a single file searching engine. To move towards database work there is a need to execute join operations, or at least to perform a quasi-join. A data management system based on the 3-schema approach has been developed. The pseudo-join is done by storing a hit bit map. However, the data must have a numeric key - if not a surrogate key is given - to ensure easy join matching. Alternatively any alphanumeric key can be hashed to give bit map position. The hardware is the FCU - file correction unit.

Notes -

(a) slower than CAFS 800
(b) but standard hardware
(c) data management system CODASYL - like in pre-compile time binding
(d) logic of searching is pseudo-relational - they have implemented in mixed hardware/low level software what INFO and G-EXEC (not RAPPORT) implement in software.

The demonstration was impressive. It took about 7 seconds to retrieve 13 records from 4K records and 6 seconds to retrieve 6 from 30K. Online update will be available in release 2 of the software.

There is some talk of attaching CAFS to ME29 as a host.

DAP: Stuart Redding

There are several new ideas on DAP:

1. Super-DAP: additional multiply unit for each PE: 4 x 8 bits plus a 128-bit register for local data.

This reduces cycle time by obviating store fetches - and also gives faster add. Expect 500MFLOP for 64 bit, 1KMFLOP for 32 bit.

Note: they have compromised the simplicity and elegance of the 1-bit PE design!

2. Nibble mode:

allows a 4 bit fetch on one cycle, and also has advantages in less wiring.
INMOS 2600 64K chip.

Mini-dap:

16 x 16 dap. 2-3 boards. 1 store, ½ logic, ½-1 for MCU

Unconstrained DAP FORTRAN:

close to ANSI ARRAY FORTRAN - proposed X8 standard.

Other things:

they are looking into the use of DAP for fast fourier transforms, number theory, design evaluation for VLSI circuits, the Queens problem and some work in data management including sorting, file searching and table lookup.

Knowledge engineering: Tom Addis

Has been working on CRIB and RAFFLES, diagnostic systems for hardware and software faults. Interested in expert systems, but problem is harvesting the expertise. Quoted MYCIN, PROSPECTOR etc. Has used CAFS.

Notes: No use of standard database system under the diagnostic system. Interesting work nevertheless, and worth watching in moves towards information systems.

Conclusions

Usual ICL problem - lot of uncoordinated research and a lot of re-inventing the wheel. Each separate area quite impressive, but no real coordination into a system for the user.

Potential use of speech output in automated office - sequence of instructions or error messages. Possible use of expert systems in information system approach.

INTERNATIONAL COMPUTERS LIMITED

RESEARCH AND ADVANCED DEVELOPMENT CENTRE

TOWARDS AN "EXPERT" DIAGNOSTIC SYSTEM

1. INTRODUCTION

(2) * Normally when consulting an "expert"

(3) * Refines the accumulated knowledge of experts so that it can be presented at anytime.

(4) * Examples of expert systems. Page 80-83

** (5) * The harvesting of expert knowledge.

(6) * But what sort of knowledge is required?

(7) * The diagnostic process - John Fox at Sheffield.

2. THE CRIB SYSTEM

(8) * The old CRIB system

(9) * Model of the machine

(10) * The process of an investigation.

(11) * Ordering the list of suggested actions. (1)

(12) * Ordering the list of suggested actions. (2)

(13) * An example

(1) 3. CAFS Storage

(2) CAFS explained

(3) CAFS (800) Facts

(4) Structured data on CAFS. (1) implication

(5) Structured data on CAFS. (2) JNF

(6) Bit maps

(7) Using CAFS for CRIB

(8) Count maps.

3. THE RAFFLES SYSTEM

(14) * The problems with CRIB

(15) * The overall process of RAFFLES - See (5) above.

(16) * The advantages of a decision tree

(17) * A Fault can be considered to be a pattern of symptons.

(18) * The pattern recognition process.

(19) * The usefulness of a symptom

(20) * Growing trees

4. THE BEST OF BOTH

(21) * Comparison between CRIb and RAFFLES

(22) * Combination of CRIB with RAFFLES

(23) * The Future

Towards an 'expert' diagnostic system

T.R. Addis
ICL Research and Advanced Development Centre, Stevenage, Herts

Abstract

'Expert' systems are computer-based systems that are capable of acquiring knowledge from experts and making it available to the less skilled. The paper describes how these particular techniques can be applied to fault-finding in computers. After giving examples of successful implementations of 'expert' systems in other fields it describes diagnostic aids being developed within ICL. Two systems are dealt with known as CRIB and RAFFLES, respectively. The paper shows how information-theory concepts can be used to improve the efficiency of the search strategy and how the ICL Content Addressable File Store (CAFS) can be used in the implementation for part of the process. The diagnostic aids described, which can be related to different elements of the diagnostic skill, will eventually be united to form an 'expert' diagnostic system of considerable power.

1 Introduction

An 'expert' system is a program and database that captures the knowledge of experts in a recognised field, makes this knowledge coherent and then available to less experienced people within that field by a man–machine dialogue. One of the important properties of such systems is the ability to provide an explanation for any given advice. One method of providing a diagnostic aid for computer systems, both hardware and software, falls conveniently within this paradigm of an 'expert' system.

Modern hardware and software are becoming more complex and are demanding a much greater level of diagnostic skill and competence from engineers and system experts. However, the availability of highly qualified and experienced personnel is becoming less as systems proliferate. Apart from developing more reliable systems, a solution to this shortage is to reduce the time spent by the skilled in finding faults. This can be achieved by providing better fault-locating aids so that either the more competent can find faults quicker or the less competent (more available) and even untrained people can locate faults which would otherwise require greater skill. Compounding this requirement for greater skill is the need for rapidly communicating new modifications of systems and fault-locating techniques gleaned from others' experience.

The first part of this paper is a brief review of some successful implementations of 'expert' systems. The second part describes work being done by ICL that is moving towards an 'expert' system for finding faults in both hardware and software. This

work is being done by ICL's Technical Services and the Research and Advanced Development Centre. It has evolved from two techniques, both of which are concerned with searching a known faults database. The first technique was developed in conjunction with Brunel University (Dr. R. Hartley) and is embodied in a system called CRIB. This system responds to described symptoms with recommended tests. The CRIB system was then modified extensively as a combined effort by Technical Services, RADC and Brunel in order to make it usable by software diagnostic specialists[1]. The other technique which is being developed by Technical Services was proposed by RADC to overcome some of the technical problems encountered with CRIB. This other technique is embodied in the RAFFLES system[2] and is based upon generating optimum search strategies. A third technique for dealing with previously unencountered faults is being explored by RADC. Each of these three systems is analogous to the three components of the diagnostic skill (see Fig. 1) and they will combine to form a commercially viable 'expert' diagnostic system.

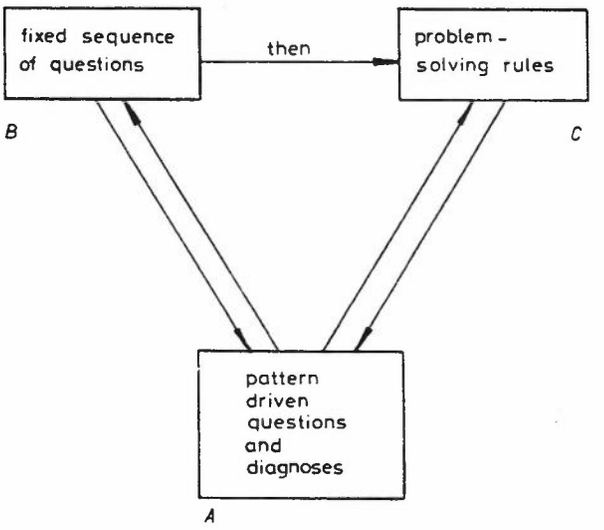


Fig. 1 Components of the diagnostic skill

2 'Expert' systems*

(Developed outside ICL.)

2.1 DENDRAL

One of the first practical systems built called DENDRAL utilised a 'production system'. This is a rule-based system where the rules are given as an unordered list of:

IF pattern THEN action.

The work on DENDRAL started in 1965 at Stanford and its task was to enumerate plausible structures (atom-bond graphs) for organic molecules, given two kinds of information:

* The section is an edited extract from Addis[3].

(*a*) analytic instrument data from a mass spectrometer and a nuclear magnetic resonance spectrometer;
and
(*b*) user-supplied constraints on the answers from any other source of knowledge available to the user.

The empirical theory of mass spectrometry is represented by a set of rules of the general form:

IF [Particular atomic configuration (subgraph) with probability, *P*, of occurring]
THEN [Fragmentation of the particular configuration (breaking links)].

Rules of this form are natural and expressive to mass spectrometrists.

DENDRAL also has an inference procedure dependent upon the Plan–Generate–Test method. The generate program called CONGEN generates plausible structures using a combinatorial algorithm that can produce all the topologically legal candidate structures. Constraints are supplied by the user and/or the 'plan' process. 'Test' discards less worthy candidate structures and rank orders the remainder. 'Plan' produces direct inference about likely substructures in the molecule from patterns in the data that are indicative of the presence of the substructure. Patterns in the data trigger the left-hand sides of the substructure's rules. 'Plan' sets up the search space so as to be relevant to the input data. 'Generate' is the inference tactician and 'Test' prunes the results.

DENDRAL's performance rivals expert human performance for a small number of molecular families for which the program has been given specialist knowledge. DENDRAL's performance is usually not only much faster but also more accurate than expert human performance and is used every day by Stanford chemists and collaborators. A program METADENDRAL was produced as an expert system for deriving rules of fragmentation of molecules in a mass spectrometer for possible use by DENDRAL.

2.2 MYCIN

There have since been many other expert systems as natural offspring of DENDRAL. The most important development was perhaps MYCIN, also produced at Stanford, whose task is to diagnose blood and meningitis infections and then recommend drug treatment. MYCIN conducts a consultation in medical English with a physician-user about a patient's case, constructing lines of reasoning leading to the diagnosis and treatment. Knowledge is acquired by confronting the user with a diagnosis with which he does not agree and then leading him back through this line of reasoning to the point at which he indicates that the analysis went wrong. The offending rule is modified or new rules added until the expert agrees with the solution.

EMYCIN (Empty MYCIN, again from Stanford) was developed to capture the framework of a general expert system. It just requires the addition of knowledge

such as the production rules. Many different systems have been created from this essential expert. An example is the signal understander (SU/X) which forms and continually updates, over long periods of time, hypotheses about identity, location and velocity of objects from spectral lines of acoustic signals. Another is PUFF which is linked directly to an instrument into which a patient inhales and exhales. From the measured flow-volume loop of the patient's lungs and the patient's records PUFF diagnoses pulmonary function disorders. There is now work going on to produce PUFF on a chip.

EMYCIN has also been used to interface an excellent but complex mechanical structure analysis package to an engineer. The engineer interacts with the expert system (SACON) to describe his problem in general terms and this generates strategy recommendations for the software package. This in turn calculates and displays the physical behaviour of the structure. Other EMYCIN derivatives are GUIDON, an expert to teach the expertise captured by the systems and AGE, an expert to guide in the construction of expert systems.

2.3 PROSPECTOR

The PROSPECTOR system is intended to emulate the reasoning process of an experienced exploration geologist in assessing a given prospect site or region for its likelihood of containing an ore deposit. The system holds the current knowledge and information about classes of ore deposits. The user begins by providing a list of rocks and minerals observed plus other observations expressed in simple English. The program matches this data against a knowledge base and requests additional information of potential value for arriving at a more definite conclusion and provides a summary of the findings. The user can ask at any time for elaboration of the intent of a question, or for the geological rationale of including a question or for a trace of the effect of his answers on PROSPECTOR's conclusions. The intention is to provide the user with many of the services that could be provided by a telephone conversation with a panel of senior economic geologists, each an authority on a particular class of ore deposits. Currently, PROSPECTOR comprises seven such specialist knowledge bases.

The drilling site selection expertise for porphyry copper deposits derives its input from digitised maps of geological characteristics and produces an output of colour-coded graphical display of the favourability of each cell on a grid corresponding to the input map. The success of PROSPECTOR depends upon the combination of clausal statements with a strong inference engine and plausible statements framed as a production system using statistical techniques (i.e. Bayes' Rule).

2.4 Comments

The advent of expert systems is perhaps the most significant event to happen in Artificial Intelligence (AI) mainly because of the huge number of potential practical applications. There is tremendous pressure from industry and the military in the USA for such systems, but there are just too few experts on knowledge engineering to provide them[4]. Oil companies are pouring large sums of money into extending programs like PROSPECTOR and variants of EMYCIN are being used for many applications by U.S.A. Government organisations.

Expert systems of the kind described in this section are required within the computer industry to interface users to complex systems and help engineers or software specialists to find their way through involved mechanisms. In particular, they can be used as diagnostic aids of considerable utility.

3 Diagnostic aid systems

3.1 Diagnostic process

Production systems (see section 2.1) can be used to describe human problem-solving behaviour. Production systems simulate a wide range of tasks from those that are stimulus-driven, like driving a car, to those that are stimulus-independent, such as mental arithmetic. Production systems can illustrate the manner in which skills are accumulated by the growth of the related production rules and the generalisation of techniques to other tasks.

In particular, the openness of the production system enables it to capture the flexibility of control that is so typical of human behaviour and so hard to capture in a non-data-driven computer program. The diagnostic skill of engineers could conceivably be represented in this form. John Fox at Sheffield University[5] has recognised three distinct components of a doctor's skill (see Fig. 1).

A doctor's actions are often made on a pattern-recognition-like basis, in which a specific question to ask (or tentative diagnosis) is suggested by a particular configuration of symptoms (A). The doctor starts the diagnostic process, however, by asking a fixed sequence of questions in order to look for an emerging pattern (B). At some point in the sequence the information is sufficient to trigger one of the rules in the pattern-driven module, perhaps leading to the asking of a contingent question, the answer to which may well trigger another of the pattern-driven rules. If this chain of rule firing does not lead anywhere the initiative reverts back to the fixed sequence of questions. If such a system fails and the doctor reaches the end of his fixed list of questions then he adopts a problem-solving strategy interacting with the pattern-driven questions and diagnosis (C).

*3.2 The CRIB System**

(A diagnostic aid developed by Brunel University in conjunction with ICL.)

3.2.1 Description: The CRIB system (Computer Retrieval Incidence Bank) was initially conceived as an aid to diagnosis for the engineer by providing him with an accumulation of symptom patterns gleaned from many sources, including his peers (this approach can be equated with A in Fig. 1). The important aspect of this collection of symptom patterns is that it is used to generate further questions, according to certain rules (heuristics), so that the engineer can quickly be guided to the fault indicated by the symptoms. It is this feedback, above all, which differentiates this kind of diagnostic aid from an information retrieval system. Thus, the engineer gives the system a description of his observations and these descriptions are compared with the database. Groups of symptoms which match the incoming description are processed to produce a list of the best possible tests for the engineer to make next.

*This section is an edited extract from Addis and Hartley.[1]

The central feature of the CRIB database is where the action is designed to elicit symptoms from the machine via the engineer. Thus the *action* coded as A1036 and meaning 'run store test program X' results in the *symptom* 'store works' or its *inverse* 'store faulty'. The symptoms can be expressed either in code (e.g. S1036 or the inverse N1036) or in jargon English as above.

The action-symptom pairs (symptoms for short) are organised in groups which can be of three types:

(*a*) Total group. The accumulated group of symptoms observed to occur during many investigations of a single fault (T-group).
(*b*) Key group. A subset of symptoms of a total group all of which are necessary to indicate the location of a fault (K-group).
(*c*) Subgroup. A subset of symptoms of a key group which, by virtue of being an incomplete key group, can only indicate the fault location to a degree less than certain.

The contents of symptom groups reflect the experience of actual investigations and a group only exists if it has proved its usefulness in actual fault investigations. In particular, there is a floating population of subgroups which are compared for usefulness by a simple success ratio (the fraction of times the symptoms in the group have occurred in successful investigations relative to the total number of investigations in which the subgroup occurred). The more successful subgroups are considered as active, whilst the less successful ones are kept in reserve.

When a group of symptoms occur together (i.e. as observed by the engineer) during a sequence of actions or tests, then this group is associated with that part of the hardware in which the fault lies. The machine is modelled as a simple hierarchy of subunits (as in a parts explosion). A simple example of the first few levels is pictured in Fig. 2.

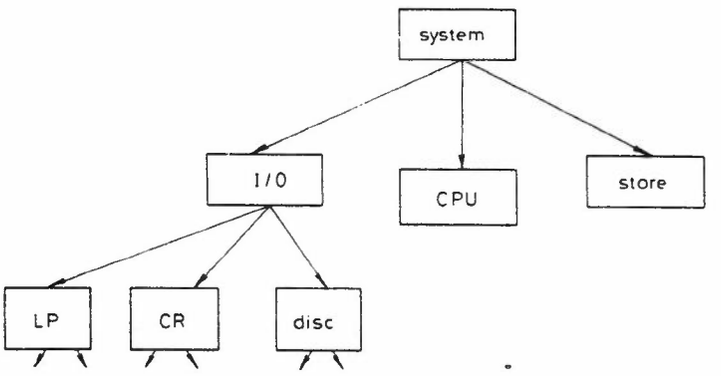


Fig. 2 A simple hierarchy of subunits of the patient computer

The point of the model is not to provide a 'true' picture of the organisation of the machine, but to provide some structure to the database so that a search can be

done in an ordered fashion. The searching of the tree is done in a depth-first fashion, where a decision to drop a level is only made on the basis of matching a complete group (total, key or sub in that order) from amongst the set of symptoms observed by the engineer. With only 50-100 nodes on the tree (subunits in the hierarchy) this is manageable.

By successively matching larger and larger symptom groups the CRIB system will eventually arrive at a terminal subunit which will be either repairable or replaceable. With the trend towards increased modularity, most fault investigations are eventually an exercise in board swapping. If a terminal subunit is reached and the fault is not cured this can mean one of four things:

(*a*) a path taken based upon a chosen subgroup did not lead to the fault;
(*b*) a key group is incomplete (more symptoms are necessary to be certain of success);
(*c*) multiple, transitory or 'soft' faults – these are difficulties in any diagnostic system, manual ones included;
(*d*) the hierarchy is incapable of reflecting the fault location precisely enough.

The system then backtracks automatically to the last decision point and tries to find another match. Since the search is depth-first the whole hierarchy can be covered if necessary, i.e. every symptom group in the database will eventually be examined. If this happens and the fault is still present then it is a fault which the system has not seen before.

When the system fails to find a match with even a subgroup at any level of the hierarchy, it asks for fresh information rather than backtracking immediately. This is not done in a blind fashion, but via a list of suggested actions put to the engineer, chosen on a heuristic basis using information about the diagnostic context current at that time. Several heuristics have been tried but all are aimed at attempting to complete incomplete subgroups thus enabling a drop in level, i.e. a more precise location of the fault. From the partially matched subgroups at that level, symptoms are compared for their usefulness according to four criteria:

(*a*) the time taken for the associated action to be carried out;
(*b*) the average amount of time the investigation is likely to take after this step has been taken;
(*c*) the number of incomplete subgroups in which this symptom occurs;
(*d*) whether the symptom is the last one in a subgroup.

These factors are combined in a heuristic procedure to order the relevant actions which have been extracted. The best five are suggested to the engineer.

The engineer is not obliged to carry out any or all of these actions, but having carried out one or more actions and having reported the symptoms he observes to the system, the search cycle is then repeated. Fig. 3 shows the diagnostic process without the backtracking.

3.2.2 Modifications to CRIB, use of CAFS: The CRIB system was constructed as described in the previous section but its testing was hampered by the lack of an efficient data-retrieval system. It was for this reason that the experimental CAFS MK I[1,6] was used which led to some interesting changes in method as well as performance.

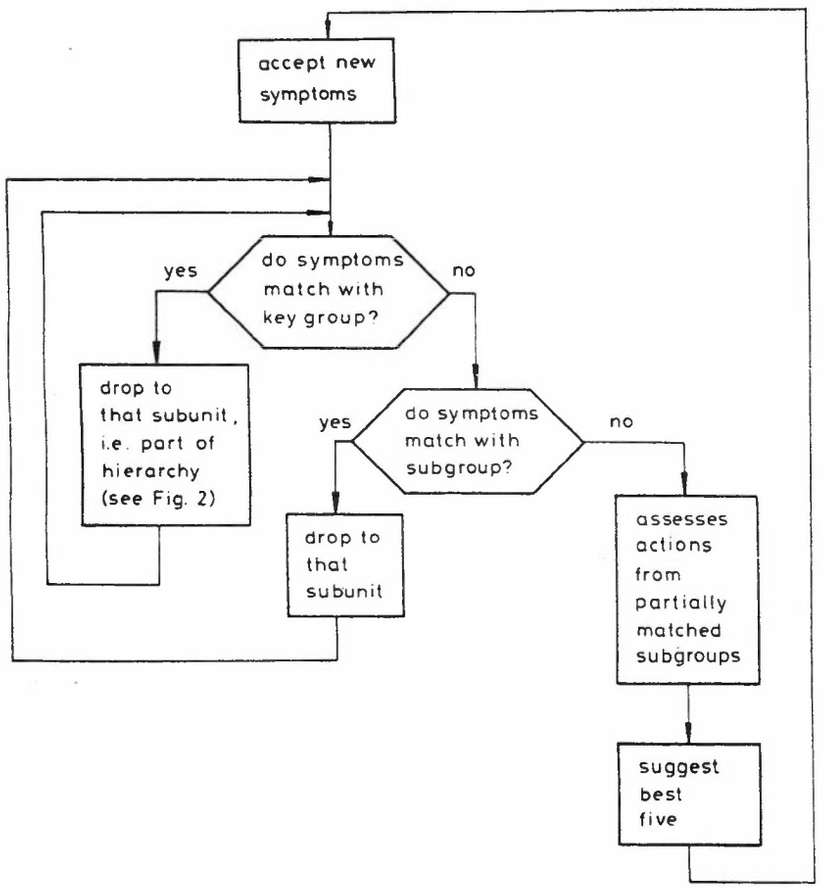


Fig. 3 Flow diagram of the CRIB diagnostic process

The early system operated as a depth-first search procedure where a match was sought only at the level below the current subunit, amongst its associated groups. Thus for all the symptom groups in the database, only those relating to the immediate subunits were examined. In this way the required search was contained.

CAFS, however, is capable of searching large databases globally, without any extra programming effort. Using this facility, the CRIB system can now look for a match with all symptom groups at all levels. The notion of a depth-first search then disappears. It is now possible for a fault to be located within any appropriate

subunit in *one* step, as against the original step-by-step, level-by-level procedure. Since this is the case, the notion of backtracking out of unhelpful situations becomes unnecessary. Because of this major change in thinking, the organisation of the program had to be altered. In particular, group matching and action retrieval techniques had to be revised.

In the original CRIB system there were four files organised according to the required major search tasks. Speed of access was limited by the need to scan these files repeatedly. On the other hand, the CAFS MK 1 system depends upon all the data existing in *one* large file where each CAFS record contains one example of each kind of data item. This means that each symptom of a particular symptom group is contained in a separate record and all the symptoms of this group are scattered throughout the file[6, 12]. Each record also contains a series of flags (known as tags) which indicate its membership of one or more of 12 virtual files. Each of these virtual files also represents the natural join of each of their implied files thus providing a considerable amount of processing precompiled[12]. Eight of these virtual files are shown as relations where the attributes that can uniquely identify an item are to the left of the colon; the attributes to the right do not suffice for unique identification.

ACTION [A:TTP,NA,LOT,DES]	- is the list of all possible tests an engineer can perform. This contains information on how long each action takes (TTP), number of times used (NA) and level of training required to perform the task (LOT). DES is a character string describing the action.
SYMPTOMS [S:A]	- forms a list of all the symptoms and observations made by engineers in performing a particular test.
TOTAL-GROUP [T:SU,CTS]	- this represents the sets of groups of symptoms associated with each subunit and are called Total groups. CTS is an attribute showing the total number of symptoms associated with each Total group.
TOTAL-GROUP-SYMPTOMS [T,S:]	- this shows what symptoms belong to which total groups.
KEY-GROUP [K:T,CKS]	- is the set of necessary and sufficient symptoms within the total groups that can isolate a replaceable unit. These are called key groups. CKS is an attribute showing the total number of symptoms associated with each key group.
KEY-GROUP-SYMPTOMS [K,S:]	- as for TOTAL-GROUP-SYMPTOMS but for key groups.

SUBGROUP [G:K,SUCC,NG,SIT,CGS] - is the set of necessary but *not* sufficient symptoms within the key groups. These are called subgroups. This also indicates the success of a particular subgroup in tracing the correct replaceable unit (SUCC), number of times used (NG) and investigation time (SIT). CGS is an attribute showing the total number of symptoms associated with each subgroup.

SUBGROUP-SYMPTOMS [G,S:] - as for TOTAL-GROUP-SYMPTOMS but for subgroups.

The CAFS MK I hardware cannot, in itself, discover whether there exists a key or subgroup which is a subgroup of the observed symptom set (more than one symptom). However, it can be done by software simulation of bit maps[7] and count maps[1]. For purposes of effective use of the maps, the target identifier values (in this case the group identifier) must be transformed into a map pointer (each bit or count word in the map corresponds to one value).

All codes in CRIB such as symptom and group identification are four digit numerals which can be used as relative addressing to a word in store. Let KI and GI refer to the four digit numeric and subgroup identifiers, respectively. CKS, CGS are the group counts, i.e. the number of symptoms in the groups. *Sx* refers to a member of the observed symptom set (i.e. one of the symptoms observed by the engineer translated into the four digit code).

The CAFS tasks*

(1) LIST KI, CKS FOR (S=Sx) IN KEYGROUP-SYMPTOMS

(2) LIST GI, CGS FOR (S=Sx) IN SUBGROUP-SYMPTOMS

are performed for each number of the observed symptoms *Sx*. The retrieval group identifiers KI and GI each point to a word in their respective count maps and add in one to the count word associated with each group found (see Fig. 4).

These numbers in the count maps, after all the symptoms have been dealt with, represent the number of symptoms observed by the engineer that are found in each group and for comparison the maximum number of symptoms possible in each group (CKS and CGS) are also retrieved (the fact that in some cases it is retrieved several times does not matter). A match is found if the number in the count map is greater than zero.

---

*The terms used are intended to be reasonably self-explanatory, but Addis and Hartley[1], Maller[6] and Babb[7] and Addis[12] may be found helpful.

If a match is found, the remaining details of subunit (and keygroup in the case of a subgroup match) are needed. There are two more tasks:

(3) LIST SU FOR (KI = Kx or Ky or . . .) IN KEY-GROUP

(4) LIST SU, K FOR (GI = Gx or Gy or . . .) IN SUBGROUP

Here Kx, Ky . . . and Gx, Gy . . . refer to those groups which have been successfully matched. It is conceivable that more than one subunit will be found, although such ambiguities are not desirable since the whole idea of a keygroup is to uniquely identify a particular subunit.

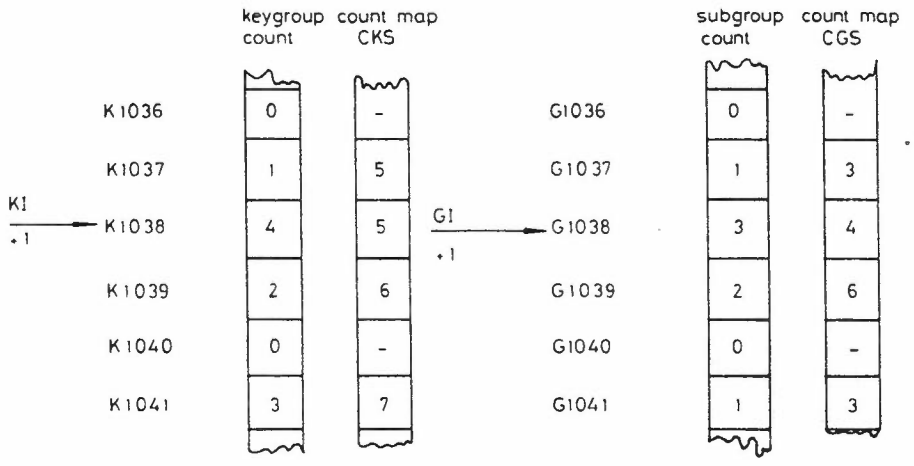


Fig. 4 Examples of count maps

To minimise the search those subgroups which are partially matched but which contain at least one symptom which is the inverse of an observed symptom, can be eliminated from further consideration. The task to do this is:

(5) LIST G FOR (S = N1 or N2 . . .) IN SUBGROUP-SYMPTOMS

where N1, N2, etc., are the inverse symptoms of the observed symptom set.

The action details for unmatched symptoms in partially matched subgroups are required for assessment. The count map used in (2) gives the partially matched subgroups (see Fig. 4); i.e. those with a count greater than zero and not equal to CGS. For each of these groups Gx, a task is needed:

(6) LIST A, TTP, DES FOR G = Gx & (S ≠ (S1 or S2 or . . .)) IN SUBGROUP SYMPTOMS

again S1, S2 . . . are the set of observed symptoms.

3.2.3 An example of the use of CRIB: Given that the CRIB system is ready for use the engineer will identify himself and then report the initial observed symptoms. This can be done either by a set of cryptic English statements or by an internal code. To illustrate the protocol consider an actual example of an engineer using the system to find a fault. The user's responses are underlined and comments are in brackets.

← LOG IN FRED, HERE
INVESTIGATION STARTED AT 10.35 ON 12.11.78

← PROGRAM LOAD FROM DISC NOT OK
SYMPTOM ACCEPTED AS NOO22. ANY MORE SYMPTOMS?

← PROGRAM LOADED FROM CR NOT OK
SYMPTOM ACCEPTED AS N0026. ANY MORE SYMPTOMS?

← ALTER AND DISPLAY ONES OK
SYMPTOM ACCEPTED AS S0402. ANY MORE SYMPTOMS?

← NO (Having accepted three symptoms, the program starts its search and makes the following responses.)

SUBGROUP MATCH FOUND IN CPU.
(Action assessment starts.)
42 SUBGROUPS EXAMINED.
9 SUBGROUPS ELIMINATED.
59 ACTIONS ASSESSED.

SUGGESTED ACTIONS:

* CHECK FOR A PARITY ERROR - A0008
* LOAD MPD WITH 2 LOG MOVES - A0415
* CHECK HAND SWITCHES INDIRECT - A0418
* RUN WAMS - A0389
* CHECK RUN LIGHT - A0413

(Here five actions have been selected as being the most effective ones from the 59 which were considered; terms like MPD, WAMS are codes which the engineer would understand.)

← SO N0008, N0415, S0413 (This illustrates the short form of reporting observed symptoms where the 'A' of the action code given is replaced by S or N depending upon the result of the test.)

ANY MORE SYMPTOMS?

← NO (The second cycle begins.)

SUBGROUP MATCH FOUND IN CPU.

68 SUBGROUPS EXAMINED (Note that there is still insufficient information for CRIB to isolate the fault any further although the search has been widened.)

39 SUBGROUPS ELIMINATED.
58 ACTIONS ASSESSED

SUGGESTED ACTIONS:

* CHANGE BOARD 9714 - A0405
* CHECK HAND SWITCHES INDIRECT - A0418
* CHANGE BOARD 9716 - A0421
* CHANGE BOARD 9710 - A0422
* CHECK STEPS OF DISC IPL ROM - A0406

LOG OUT OK

(In this case the first suggested action cleared the fault. The engineer was satisfied the diagnosis was complete and terminated the run as OK.)

INVESTIGATION FINISHED AT 10.40 ON 12.11.78
DURATION OF INVESTIGATION 4.5 MIN.

3.2.4 Assessment of CRIB: There were several difficulties which arose out of the use of CRIB. The first problem was perhaps the tendency of the CRIB system to spread its considerations over more and more groups every time a symptom was added. This had the double effect of, first, slowing the response because of the greater number of tasks to be generated and second, the insertion of apparently irrelevant actions into the suggested list because of it making contact with distant groups. This problem really reflects the method used in the CAFS search and could have been contained by using the hierarchical model (see Fig 2) to constrain the search to the relevant part of the machine under consideration.

A second problem was the uncertainty of how to answer some actions to select the desired symptom. This problem falls into two distinct parts.

The first is the problem of the jargon English interpreter which, though perfectly adequate when dealing with a handful of contextually dependent sentences, was found to be incapable of handling a large range of open-ended descriptions. The engineer could never be sure either how to phrase a sentence so that it would be accepted or if it was accepted whether it had been identified correctly. However, users of the system quickly learnt a few starting sentences and then employed the returned action code with the appropriate prefix to continue. The second part of this problem was that some answers to an action were ambiguous. If a symptom was observed there could be uncertainty as to whether a positive or negative response to the suggested action was expected.

Another (third) problem, which is really a consequence of the second, was that there was no way in which a set of symptoms in a new group (indicating a freshly discovered fault) could be checked to see if they already existed. Trial and error, or scanning down all the existing symptoms or using some kind of selector mechanism based upon the concatenation of words in a sentence (keywords) might help, but they are all uncertain methods of checking. There was also the related update

problem in that no guarantee could be given that there existed a distinguishing symptom in each group that would ensure isolation of a fault. This last problem could be overcome by employing the diagnostic techniques during update.

Despite these difficulties (most of which have solutions) the system proved to be successful and in particular the suggested actions helped the engineer to consider areas of investigation which he would otherwise have overlooked. This had the useful result in that even if the fault being investigated was *not* to be found within the CRIB system it could still help the engineer isolate this new fault by suggesting avenues of enquiry.

3.3 A Simplified CRIB

3.3.1 Description: Shortly after the CRIB (CAFS MK I) system was implemented there was the suggestion that it could be used for system software investigations. In particular it could be used for VME/K and DME known faults by acting as the initial pre-scan before technical specialists were involved. With a considerable amount of help from those technical specialists and after several variants of CRIB and its database had been tested a simplified version of the CRIB system was constructed.

The concept of subgroups was abandoned, leaving just total and key groups. Each action was given a priority ranking instead of 'time to perform' (TTP) and marked either *P* (for patch) or *Q* (for question). Each key group (now called Trace) was given a 'likelihood' which acted as a priority modifier for all the actions within the key group. This priority was used as one of the factors in ordering the suggested list of actions returned to the user. The user then had the choice of the following: *either* having the returned suggested actions ordered according to the number of hits and (for those with the same number of hits) ordered according to the priority modified by 'likelihood' *or* having the suggested actions ordered according to the modified priority only.

The user also had the choice of listing out any number of the ordered actions he wished instead of just the first five. 'Likelihood' was intended to indicate the probability of a symptom group occurring so that as the database is updated so this parameter would be adjusted for each group.

The problem of restraining the CRIB system from contacting distant groups was left in the hands of the user by providing him with two distinct selection mechanisms. The first mechanism used the count maps in three ways:

(*a*) If the user has reported m symptoms ther. he can select from the count map only those groups which have a count greater or equal to $(m - n)$ where n is an integer of his choice. If n is zero then this is equivalent to insisting that *all* the symptoms given are contained in the groups retained for consideration.

(*b*) Irrespective of how many symptoms the user has reported the system can scan the count map looking for the maximum number of hits and only

reporting on those groups with this maximum number. A simple extension of this idea is to allow (Max - n) as the selection mechanism.

(*c*) Count map selection can also be done relative to the total number of expected symptoms in a group (CKS in Fig. 4). The selection in this case is done by subtracting n from each group number (CKS) and if the count is greater than or equal to this then it is returned. Another way of describing this method is that it selects those groups which have at least a specified number of symptoms in total. Smaller groups than this number behave as (*a*), but the smaller the group the more symptoms are ignored. The number of symptoms given by the user + n = maximum total in groups.

The second mechanism employs a modification of the unit-subunit hierarchical structure used in CRIB (Fig. 2). In this case the hierarchical structure represented an organised approach to diagnosis constructed from the experience of the experts, where each level in the hierarchy represented sets of questions concerned with details of the level above. A five-figure code was constructed so that each character position represented a level and the character chosen gave the branch at that level. Fig. 5 shows part of this structure where the code at each level is given in brackets and each level is associated with a priority number (the lower the number the higher the priority). The user is then able to confine his search to any subsection of this hierarchy by typing in for example SU 32B** or SU **B*D where *represents a 'don't care' character.

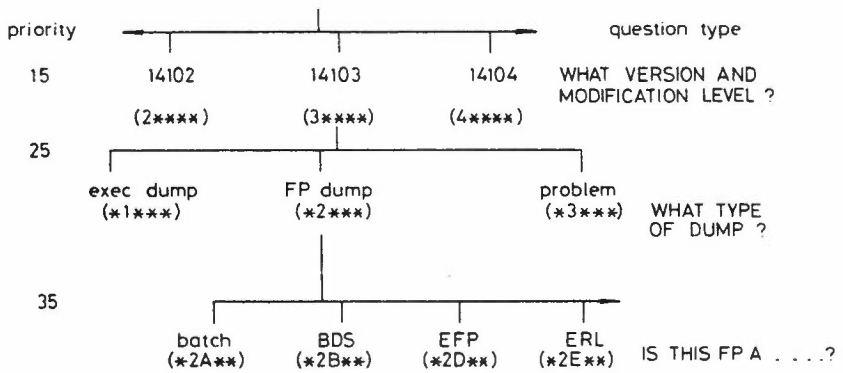


Fig. 5 Part of the VME/K faults database structure

The problem of ambiguous answers to actions (questions) was solved by simply allowing only one answer to be accepted - namely 'YES'. Thus the question 'is operating system version 14?' would either be answered 'YES' or not answered at all. This decision to have only one answer leads to other problems, but these other problems are easier to cope with for this application than ambiguity.

3.3.2 Assessment of simplified CRIB: The simplified CRIB is more of a tool in the hands of experts than the original since many of the strategies of pattern matching are decided by the user. The speed of response was considerably improved and the

probability of success greatly enhanced with most of the original problems bypassed. The only outstanding problem was that of ensuring that symptoms were not repeated in another form when updating occurred.

This problem was partially overcome by classifying symptoms. Each class then consisted of a sufficiently small number of symptoms to be scanned by eye. It was intended to make an extended trial of this modified system but conflicting demands made it necessary to curtail this; however enough was done to show the importance of involving, at quite a detailed level, the experts and the end users in the development of such a tool. The final system required less than five minutes explanation to an end user for him to be using it efficiently. *The importance of such a tool as this is that it represents a medium by which an expert can express and record his expertise in a way that is self maintaining.* This is why it requires the specialist to create the database.

3.4 The RAFFLES system

(A fault-finding system currently being developed at Stevenage for use by Customer Services.)

3.4.1 Description: The CRIB systems were designed for a single user and depended upon fast data retrieval and processing, giving maximum control to the user. The objectives of RAFFLES, on the other hand, were to make a diagnostic aid available to a large number of people simultaneously without the need for expensive processing. As has been suggested in Section 3.1, and from observations of users of CRIB, the diagnostic sequences tend to be highly constrained. This results in much of the online working of CRIB being continually repeated at a time when minimum processing is required. Other problems of CRIB have been interpreting the symptoms (expressed in jargon English) and the control of symptom updates (see Section 3.2.4).

These problems can be overcome by the proposed Rapid Action Fault Finding Library Enquiry Service (RAFFLES) which can be considered as a compiled version of CRIB but with a test selection scheme based on information theory rather than heuristic methods. Since all the decision making is done offline this means a very simple online program. This scheme is designed to draw together the mass of different test procedures used and discovered by many engineers into a unified approach to fault finding. The system comprises principally two programs: Generate Tree and Display Tree as shown in Fig. 6. Generate Tree generates the optimum fault location procedure in the form of a multibranching decision tree. Each node of the decision tree is a test and each branch a result of that test. The decision tree is interpreted by displaying the tests at the top of the tree and according to the result provided by the engineer steers him down to a terminating node which represents the unit to be replaced. RAFFLES employs a technique which was successfully used for pattern recognition.

The RAFFLES system should have the advantage of being extremely fast; requiring minimum computer power, since all the hard work of selection and scanning is

done offline by Generate Tree. The selection of test criteria is based upon the well understood and tried techniques of information theory which provide the fastest test sequences for fault location (which can also be issued in book form using program Print Tree), expose the existence of any equivalent or fallacious tests and avoid sequences of similar tests. The need for jargon English input becomes de-emphasised but is still capable of being supported when available, and the interactive protocol provides an ideal system for using speech as a means of communication. However, as with CRIB, the effectiveness of the RAFFLES system in locating faults depends entirely upon the quality of the data digested by the Generate Tree program. This will accrue over a period of time from actual results in the field as the file accumulates via program UPDATE (see Fig. 6).

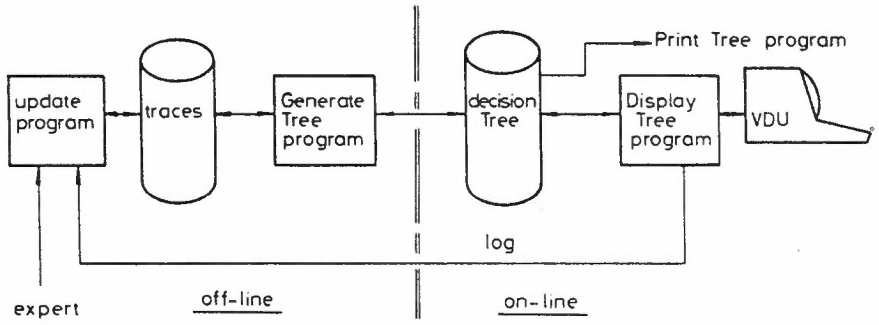


Fig. 6 The RAFFLES system

3.4.2 Theory of the Generate Tree Program: The objective of the Generate Tree program is to sift through examples of successful fault location traces (e.g. groups of symptoms that lead to a fault) associated with each replaceable unit and generate an optimum fault location guide in the form of a multibranching tree. The first node of the tree represents the best test which divides the classes into as near to equal-sized and stable groups as possible.

The second set of tests in the next level down the tree repeats this requirement for each divided group and so on until the final test in a branch isolates a replaceable unit. The measure of 'best' for a test requires that the test is both reliable and provides an equal division between the classes. In information terms, this represents the test that brings about the greatest reduction in relative entropy at each level in the hierarchy and in this case, where tests can take varying lengths of time, the greatest reduction in relative entropy per unit time.

The faults associated with each replaceable unit (or fault report in the case of software) can be considered as a series of independent symptoms that identify that unit. A trace is an actual example of a fault identification. Each symptom represents the result of an action where each action (usually a test in the form of a question) can have one or more possible outcomes (answers). The effect of dividing the database of traces (symptom groups identifying a fault) according to the result of any particular test is to reduce the entropy of the whole system with respect to the

faults. That is, the faults become partially ordered. The best test will be the one that reduces the entropy by the greatest amount in the shortest possible time.

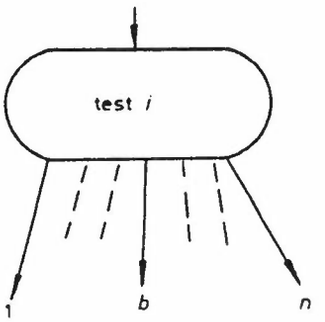


Fig. 7 Test i with n associated symptoms

The determination of such a test can be achieved by considering the statistics of each test when used without any constraints, such as in the CRIB system. Let the range of tests be $1 \leqslant i \leqslant T$, the range of possible outcomes of test i be $1 \leqslant b \leqslant n$ and the range of possible faults, or of replaceable units in the case of hardware, be $1 \leqslant k \leqslant K$.

We define the following probabilities:

$p(k/b, i)$ = probability that fault k is present (or unit k is faulty) given that test i has outcome b.

$p(k, b/i)$ = probability that test i will have outcome b and that fault k is present

$p(b/i)$ = probability that test i will have outcome b

$p(k/i)$ = probability that test i will give the conclusion that fault k is present.

It can be shown that given any outcome b for test i the entropy with respect to the set of K possible faults (or replaceable units) is

$$H_{b,i} = -\sum_{k=1}^{K} p(k/b, i) \log_2 p(k/b, i).$$

Then the expected entropy $E(H_{0,i})$ of the outcome, given the test i, is

$$E(H_{0,i}) = \sum_{b=1}^{n} p(b/i) H_{b,i}.$$

This becomes after suitable manipulation

$$E(H_{0,i}) = -\sum_{b=1}^{n} \sum_{k=1}^{K} p(k, b/i) \log_2 p(k, b/i) + \sum_{b=1}^{n} p(b/i) \log_2 p(b/i).$$

The input entropy (i.e. before the results of any test are known) is

$$H_{I,i} = -\sum_{k=1}^{k} p(k/i) \log_2 p(k/i).$$

The expected decrease in entropy $E(\Delta H_i)$ resulting from applying test i is $H_{I,i} - E(H_{0,i})$ and from the above equations this is

$$E(\Delta H_i) = -\sum_{b=1}^{n} p(b/i) \log_2 p(b/i) + \sum_{k=1}^{K} p(k/i) \sum_{b=1}^{n} p(b/k,i) \log_2 p(b/k,i)$$

If the test i takes t_i seconds to perform, then $E(\Delta H_i)/t_i$ is the entropic decrement per unit time.

Test i is chosen such that $E(\Delta H_i)/t_i$ is maximum for the set of traces under test. This having been chosen, the next set of tests is calculated independently for each of the resulting divisions of these traces caused by applying test i.

A full description of these calculations is given by Addis[2].

3.4.3 Using decision trees: The process of creating the decision tree is now simple. A database of actual test sequences that led to the location of a fault is analysed according to the above formula. This will provide every test with a measure of excellence in the form of the entropic decrement per unit time $E(\Delta H_i)/t_i$. These can then be ordered according to this measure of excellence and this ordered list of tests is now equivalent to the initial ordered list of actions that can be printed by CRIB. However, the Generate Tree program forces the issue by taking the test with the highest value (the test at the top of the list) and proceeds to use that test to divide the database of traces into n possible groups, n being the number of possible outcomes of the test. Fig. 8 illustrates this process.

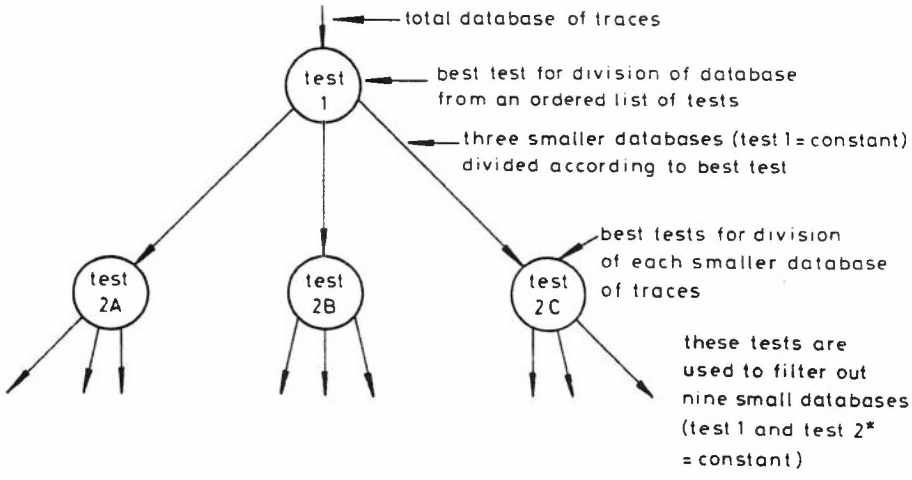


Fig. 8 The Growing Tree

There are three useful byproducts of this technique which resolve a problem with the CRIB system. The first is that symptoms which carry the same or similar information irrespective of how they are expressed will have their measure of excellence reduced in sympathy with the chosen test. Suppose test z is chosen as the next test along a certain branch of the tree instead of say tests x or y. After the database has been divided according to that test, and if it is found that test x which had formerly a high entropic decrement value is now low, then it is reasonable to conclude that x and z are equivalent tests and should thus be combined. The second useful byproduct is that if a branch of a tree is reached where there are no more possible tests to distinguish between faults then this will be discovered and an appropriate request can be made to the overseeing expert for a distinguishing test.

The third byproduct is that tests which carry little or no information can be discovered and discarded.

The method of selection of tests in creating the tree does not take into account any other factors which might influence the choice of test. Certain tests may be too difficult for the user to perform, or too trivial to be worth considering because the general fault location is obvious, or just not possible because they depend upon certain equipment. Further, the engineer may just have additional knowledge of this particular environment which would guide his choice.

Some of these problems can be overcome by creating different kinds of trees for different purposes and by providing a mechanism for using the trees effectively with inadequate information. The existence of a test at any point in the tree is totally dependent upon the sequence of answers to the tests that led to it through the tree. A test out of context has little or no meaning to the system. However, there are at least three possible strategies which can resolve this problem.

(*a*) It is possible to assign each test to a classification representing the skill required to perform that test. Different trees can be created according to the level of skill of the users by selecting only those tests that are within an ability range.

(*b*) The 'patient' mechanism can be considered to be constructed in some hierarchical fashion as for CRIB (see Fig. 2). If this is done then a tree can be grown which calculates the entropic decrement with respect to a level in this hierarchy instead of the final replaceable units (or fault document).

Fig. 9 illustrates a simple tree that decides the general area of the fault down to the first level. Once the general area has been selected another tree can be grown which calculates the entropic decrement with respect to the next level in the hierarchy. This is continued until a set of trees is formed that isolates all the faults (the lowest level in the hierarchy).

The advantage of such a system is that any expert user can jump *directly* to the appropriate level in the hierarchy thus bypassing the intermediate stage of answering the often tedious set of questions that would get him there. Each tree is constructed with the assumption that the user has correctly classified the general area

of the fault. The method by which this classification has occurred is irrelevant so that any tree grown from that point has tests dependent only upon this correct classification. The disadvantage is the extra work required by the Generate Tree program to produce these different trees and also that the total number of tests to find a fault is likely to be more if a user were to start from the top level. However, this second point can be avoided by providing trees at all levels that home in on faults directly.

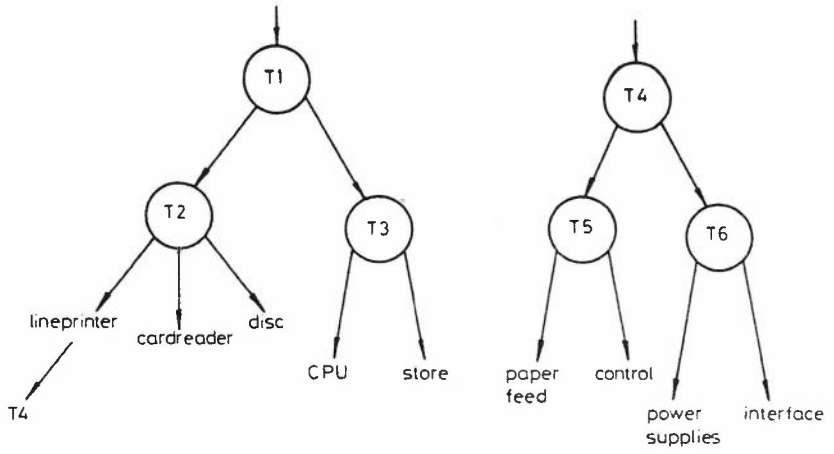


Fig. 9 A top and second-level tree

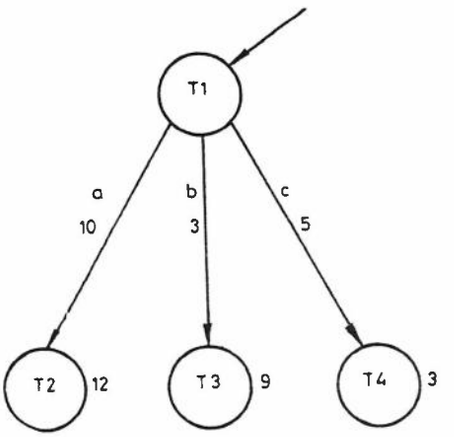
(*c*) When a question, given to a user from the RAFFLES system, is answered by 'Don't know' a problem arises in that the RAFFLES system then has to simultaneously progress through every possible answer for a given tree. If other questions are subsequently answered by 'Don't know' then this will create a very large number of simultaneous paths to be traversed. The management of such a multitude of traversals becomes extremely difficult.

A solution is *not* to traverse the paths simultaneously but consecutively choosing the most-likely-to-succeed path first. If a solution is found before all the paths have been traversed then the process can halt. This process requires that every answer to a question at a particular node has an order of precedence indicating which answer will lead to the most likely conclusion.

This order is directly related to the entropy associated with each branch H_{bi} (see Section 3.4.2). The greater the change in entropy of a branch the higher the ranking. The highest ranking answer which has *not* been tried before is automatically tried and marked. It is interesting to note that this gives a similar result to the CRIB heuristic methods in that any branch (symptom) that leads to the isolation of a fault (or level) will always give the greatest change in entropy. This is because the entropy will fall to zero. Consequently, as for CRIB, the last symptom in a group will always be suggested for testing before other routes are tried.

The user can then be given the option to continue until the final node of a particular path *or* backtrack to the last (last-1, last-2 etc.) marked node *or* both. The user can be given control to re-enter a path that was abandoned before the final node was reached with the option to continue (with perhaps a summary) from where the path was previously abandoned.

Rather than keep the user completely in the dark the next group of tests beyond the one under consideration can be presented in an ordered list according to the reduction in entropy taken to reach them (see Fig. 10).



T1

ΔH_b	ans	$E(\Delta H)/E$	Next test
(10)	a	(12)	T2
(5)	c	(3)	T4
(3)	b	(9)	T3

Fig. 10 A look ahead table for the end user

So far the assumption has been that every test provides a few distinct outcomes which can be represented as a branch in a tree. In principle any test can be broken down into a sequence of primitive tests with a binary outcome. If this is done then the problem of answering such sequences can be achieved by running these primitive tests as a program. For example, if a computer store address is required and the system is aware of a set of possible computer addresses then it would not be reasonable to expect the user to answer all the 'is the address = x' for every value of x. The principle of the Generate Tree program can still be applied to this sub-task so that the questions can be asked in the most efficient order with the user supplying only the address value (see Fig. 11).

Some results of tests can be potentially infinite but the range is limited. Such tests are the measurement of some voltage, current or length. In these cases the entropy of the system becomes a function of the result (see Fig. 12). The choice of tests to follow may depend upon threshold values of the function.

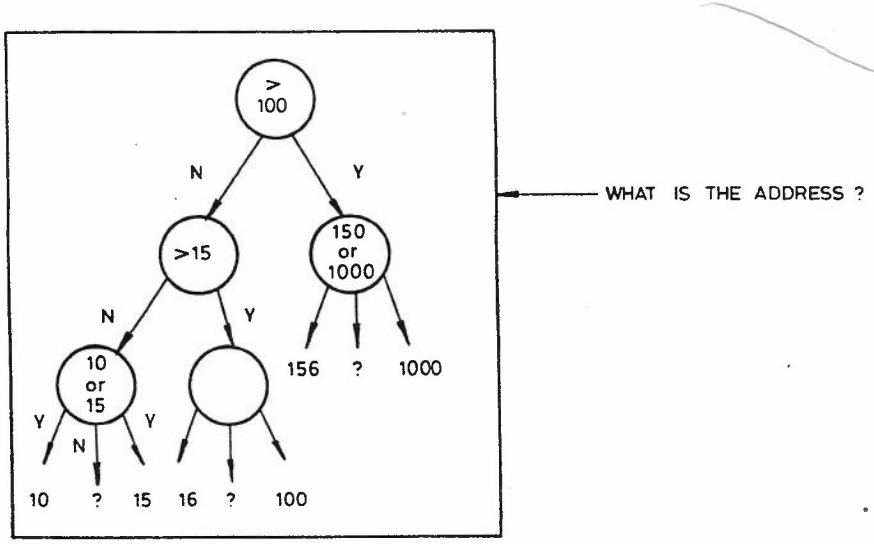


Fig. 11 A single complex test as a set of primitive tests

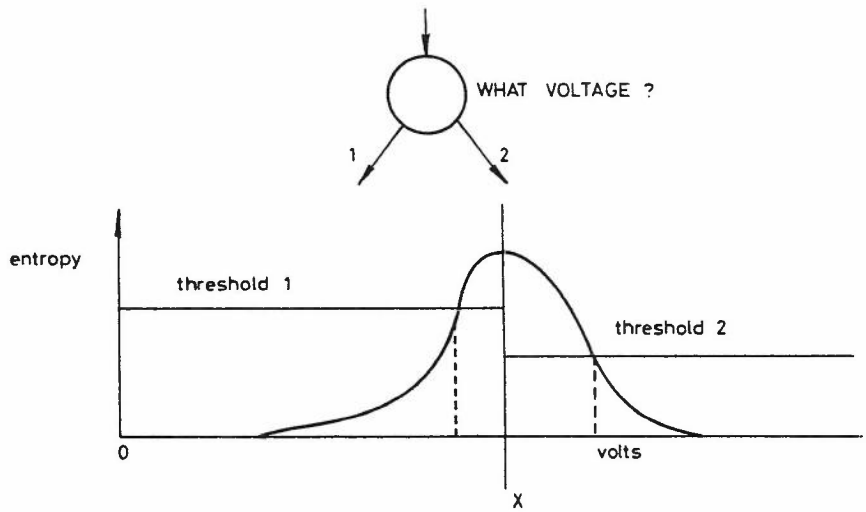


Fig. 12 A continuous function of entropy

It is important to note that any expert providing a sequence of tests for isolating a new fault for the RAFFLES (or even CRIB) system should choose tests which are going to provide a hierarchical division of the problem.

If most of the tests are too general or specific, then the Generate Tree program will generate a deep and imbalanced tree. Experts updating the diagnostic system should be made aware of both what is required and how their particular additions are assimilated.

3.4.4 Combining RAFFLES and CRIB: There are two problems with the

RAFFLES system which are not found in CRIB. The first problem is that the user is driven and constrained by the tree. This means that any extra knowledge about the problem the user may have cannot be fully utilised by the fixed diagnostic procedure. The advantage of this fixed procedure is that it sidesteps the difficulties of 'natural' language interpretation (see Section 3.2.4). The second problem is that towards the terminating tests in the tree, the statistics upon which the entropy measure is calculated start becoming unreliable because of the fewer number of examples upon which they are based. However, CRIB does not suffer from either of these difficulties because any errors due to statistical anomalies are adjusted by the common sense of the user. On the other hand, CRIB requires a considerable amount of processing power to provide this flexibility.

The solutions to these problems are not incompatible and it is possible to achieve a balanced solution by combining the advantages of both systems and reducing the disadvantages. Most of the CRIB processing is due to the quantity of data that has to be sifted and compared. One of the properties of a RAFFLES tree is to divide a very large database into many small databases at each level. So if a RAFFLES tree is constructed up to the point where the statistics become unreliable then this will provide many small databases each of which can be scanned with much reduced processing by the CRIB system. This combined system is illustrated in Fig. 13.

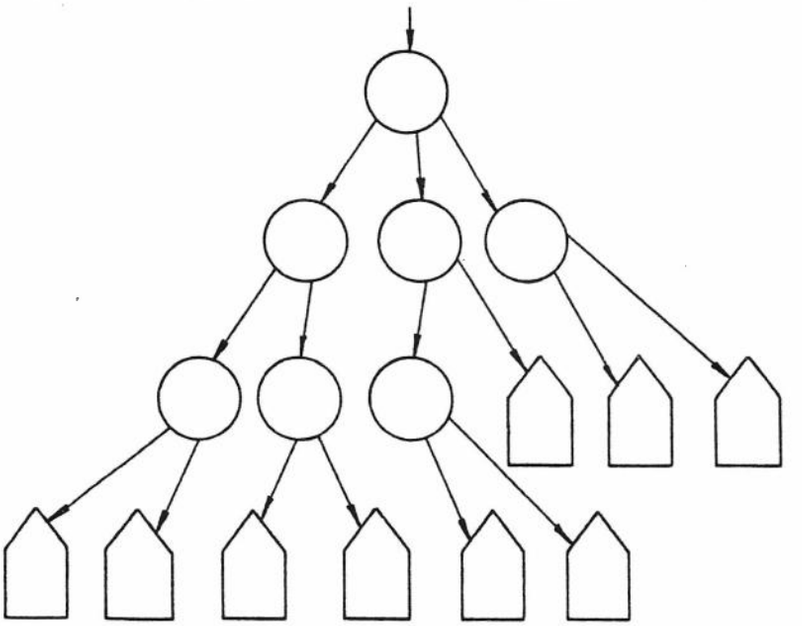


Fig. 13 Using a tree to create many small databases

3.4.5 Comment: In practical terms RAFFLES provides a predigested faults database accessed by disc. However, unlike any normal retrieval system it provides tests for the user to perform which have a high probability of success; the database is self maintaining and the online interactive program (Display Tree program) is simple.

3.5 *New faults problem*

RAFFLES and CRIB are concerned only with those faults that have been encountered previously. It is estimated that 30% to 50% of faults investigated in software and hardware are previously unencountered faults. It would be desirable to complete the diagnostic system by providing an aid for isolating these new faults. The difficulties, when anybody is faced with determining why a mechanism is not working, are what tests should be performed and what current observations are relevant. Previously, for encountered faults, this was simply repeating what had proved successful in the past. It now becomes necessary to generate new tests and make fresh observations in the light of what is already known.

For a system to generate tests on a mechanism (or to test the 'correctness' of current observations) requires that there is some representation (model) of the component parts of the mechanism; how these component parts interact and what happens when a component part fails in a particular way. Progress is being made on representing the medium to large range of machines down to gate level in order to generate diagnostic test programs[8]. This kind of precision, although leaving nothing to chance, can be much too detailed for a general fault finding guide. (However, this kind of detail is required for the automatic generation of test programs.) A more promising approach is a hierarchical description of the mechanism at the appropriate levels of coarseness. This description should be biased towards the task of fault finding rather than machine emulation.

A method for describing and running mechanisms in the abstract is currently being researched (known internally as the Broadside Programming project). This method is based upon principles developed by those involved in 'natural' language understanding by machine[9] and is currently in the feasibility study stage; whilst in this state of examination no details will be given in this paper. Machine understanding means that interpretation of any sentence, paragraph or story does not depend entirely upon grammatical constructs, but relies upon a working representation of what is being said in the light of previous accumulated knowledge. The difficulty of developing such representations depends upon choosing the correct elementary building blocks and primitive processes that can cope with a wide range of descriptions. Language analysis has shown that such elementary units exist but these units have to be modified quite extensively to fit our particular requirements of modelling mechanisms.

There are two provisional conclusions that can be drawn from this work. The first is that the concept of high and low-level languages vanishes. The objects manipulated by a high-level language are just different from those of a low-level language. Level, in fact, lies outside the language domain altogether since it describes the relationship between two sets of objects via an intermediary. The second is that there is no definable entity called 'natural' language. What comes to be called 'natural' language actually represents a pot-pourri of many different languages each of which is a highly honed tool for a particular task. The reason for the confusion is that there is a strong family resemblance between these languages since they were born out of the way man perceives reality.

This work, although only just begun, can expect to resolve the last of the problems experienced with the diagnostic system. It should now become possible to define a diagnostic language which will have some features of 'natural' language. This language would be used not only to describe the mechanism, but also to report symptoms. The products of the system (supporting such a language) will be useful views of the mechanism given by expert fault finders, which would be used for both generating suggested tests and checking the consequences of current observations.

4 Conclusions

This paper has described the 'potential' evolution towards an 'expert' diagnostic system. It is 'potential' because it has not yet been fulfilled. The CRIB system has been implemented and forms the basis of the pattern-driven-question-and-diagnostic component of the diagnosis skill (see Fig. 1 Section 3.1). The RAFFLES system is under construction and represents the fixed-sequence-of-questions component of this skill giving the most efficient fault finding guide. The NEW-FAULTS system is being researched and provides the problem-solving aspect still missing.

Independently, each of these three systems provides unique benefits to the user but they also have defects that are cancelled out by the existence of the other two systems. The defects with CRIB were essentially the need for describing symptoms so that they could be uniquely identified and the large processing requirements necessary for determining the next step. RAFFLES, on the other hand, cancels out all these problems by side-stepping the user symptom descriptions by printing out questions to be answered, identifies similarity of symptom descriptions on the database by their entropy behaviour and avoids doing the processing during diagnostic time. However, it loses the flexibility of CRIB that allows the experience of the user to be used to good effect. By judicially combining CRIB and RAFFLES, flexibility is to some extent restored without an excess of processing. The NEW-FAULTS system will probably also need a considerable amount of computation but it should only be required part of the time for NEW-FAULT resolution. The additional gain of the NEW-FAULT system is that it will provide a technique for describing symptoms uniquely and give lines of reasoning for any particular diagnosis*. The combined systems of CRIB, RAFFLES and NEW-FAULTS should interact to form a complete 'expert' diagnostic system.

The important difference between an 'expert' diagnostic system and that of a database retrieval system is that it provides guidance to the user, it has a database that is self-maintaining and it provides a medium through which expertise can be expressed. Without the 'expert' there can be no 'expert' system.

---

* It is interesting to note that such a complete system would not always be able to 'explain' why a certain sequence of tests will produce the desired result because the source of such information would come from the accumulated knowledge obtained from experts' experience. This is equivalent to any expert's 'rule of thumb'; knowing that a certain sequence works without knowing why.

5 Acknowledgments

Since this work was initiated by A.H. James (TS) and Prof. F. George (Brunel University) there have been many people who have provided invaluable help for this project. Thanks should be given particularly to Dr. R. Hartley who did an excellent job under difficult conditions to transfer his CRIB software to work with CAFS (Content Addressable File Store); D. Icke and his team of software specialists who worked overtime to capture their skills within the CRIB system and showed us clearly that you cannot have an expert system without the experts; A. Ingram and his team who are currently involved in the difficult task of making practical the theoretical concepts of RAFFLES; and G. Piper who has managed to make this co-operative venture work within an ever changing environment.

References

For the sake of completeness references to a number of ICL internal notes and reports are given. Not all of these are available outside the Company; any reader who would like to consult one of these is invited to write to the author.

1 ADDIS, T.R. and HARTLEY, R.T.: 'A fault-finding aid using a content-addressable file store'. Technical Note TN 79/3 June 1979.

2 ADDIS, T.R.: 'A feasibility study of RAFFLES' Technical Note TN 78/3 August 1978.

3 ADDIS, T.R.: 'Report on the AISB Summer School on expert systems in the micro-electronic age'. Internal Note TRA. 79/6 July 1979.

4 FEIGENBAUM, E.A.: 'Themes and case studies of knowledge engineering'. Published in 'Expert systems in the micro-electronic age' (Proceedings of AISB Summer School), Ed. Michie, D. Edinburgh University Press.

5 YOUNG, E.M.: 'Production systems for modelling human cognition'. Published in 'Expert systems in the micro-electronic age' (Proceedings of AISB Summer School), Ed. Michie, D. Edinburgh University Press.

6 MALLER, V.A.J.: 'The content addressable file store - CAFS' *ICL Tech. J.*, 1979,1 (3), 265-279.

7 BABB, E.: 'Implementing a relational database by means of specialised hardware'. *ACM Trans. Database Syst.* 1979, 4 (1), 1-29

8 CORRIN, P.G. and McLAREN, K.: 'System diagnostics – achivements and future work, July 1979'. STDC Information Note SOFTECH/IN/500, issue 0.

9 ADDIS, T.R.: 'Machine understanding of natural language' *Int. J. Man-Mach. Stud.*, 1977, 9, 207-222.

10 SHANNON, C.E. and WEAVER, W.: *Mathematical theory of communication*, Urbana, University of Illinois Press, 1964.

11 FU, K.S.: Sequential methods in pattern recognition and machine learning. London and New York, 1958, Academic Press.

12 ADDIS, T.R.: 'A relation based language interpreter for a CAFS'. Technical Report 1209 August 1979.

Giving the computer a voice

M.J. Underwood

ICL Research and Advanced Development Centre, Stevenage, Herts

Abstract

Recent developments in semiconductor technology, together with advances in digital signal processing, have led to silicon chips that talk. Important though the techniques for speech generation are, their successful and widespread use in commercial computing systems will depend upon their careful incorporation into the overall systems design. This paper describes a systems approach to giving the computer a voice that will enable the end-user to obtain information easily and accurately. The first part of the paper is concerned with an analysis of requirements, with particular emphasis on the needs of the end-users. The concept of a speech subsystem as a modular component is described, together with some details of its prototype implementation. Some indication is given of the current and future applications for a computer with a voice. Spoken output from computers is likely to play a more important role in the future as the pattern of computer usage moves towards more human-oriented information processing.

1 Introduction

Speech has evolved as man's most natural and important means of communication. Spoken communication developed much earlier than the written form, yet when it comes to the communication between man and his information-processing artefact, the newer written form is dominant. Why should this be? The answer probably lies in the different nature of the two means of communication. The evolution of speech is deeply interconnected with the evolution of human behaviour as we know it to-day. Organised human life without speech is inconceivable. Because it is so much a part of us it may be very difficult for us to be introspective about it and to understand it fully. As a means of communication it is informationally much richer than writing. Writing can be be regarded as a sub-set of natural human communication and this has led to it being mechanised much earlier. The ability to generate speech, as opposed to transmitting or recording it, has had to await the arrival of the information processing age.

There are two aspects to spoken communication, its generation and its understanding. Eventually the techniques for the automatic execution of both of these processes will have been developed to the point where people will be able to communicate naturally with computers; it is likely to be many years before this is achieved. Indeed, it could be argued that such a situation is unnecessary and undesirable, since the reason for designing information processing systems is to complement

man rather than copy him. Nevertheless, as more people come into contact with computers in their everyday lives there is a need to improve the means of man-machine communication. This paper is concerned with the requirements for giving the computer a voice and with the solution that has been developed in the ICL Research and Advanced Development Centre (RADC). The aim has been to design an intelligent speech controller that provides the necessary facilities for speech output in the commercial computing environment. The recently announced semiconductor 'speaking chips' have been designed for different purposes to meet mass-market requirements.

2 Speech as a communication medium

As a means of communication, speech has both advantages and limitations and it is important to understand these before attempting to design speech output equipment.

First, the advantages, as these will play an important part in determining how speech output is used. Speech is an excellent medium to use for attracting a person's attention, even though their eyes are occupied with another task: it is very difficult to block the auditory channels. It is excellent also for broadcasting information simultaneously to many people as it does not require direct line of sight. It can also be a very private means of communication if played via a headset directly into the listener's ear. Finally there are very extensive speech communication networks in existence – the national and international telephone networks – and it would be very attractive to be able to use these for computer-to-man communication without the need for equipment such as modems. Every telephone would then become a computer terminal.

An important limitation is imposed by the fact that the rate of speech output is controlled primarily by the speaker and not by the listener, which means that, unlike printed output, speech output cannot be scanned. Therefore computers, like people, should talk to the point. Another arises from an interesting property of the human processing system, the restricted capacity of the short-term memory, typically seven items.[1] This places a limit on the amount of information that can be presented to a listener at any one time if he is to remember it well enough to do something useful with it, such as write it down. A further characteristic is that speech leaves no visible record and therefore is best suited to information which is rapidly changing or ephemeral.

The main conclusion to be drawn from this is that speech should not be regarded as a replacement for existing means of machine-to-man communication but as a complementary channel, best suited to the transmission of certain types of information.

3 Requirements

The requirements of three groups of people have to be considered: the user (i.e. the listener), the programmer and the equipment supplier. The primary objective of the user is to obtain information easily and accurately, whilst the programmer is

concerned with what the machine is going to say and how. The objective of the equipment manufacturer is to supply and support the requirements of the listener in a cost-effective manner. The requirements of these three groups affect not only the choice of speech output technique but also the way any speech subsystem is interfaced and controlled. In addition, the use of speech may have implications for the design of the system which uses it. For example, the information held on a file may need to be different to allow spoken output as opposed to printing.

3.1 Listener's requirements

The way in which the information is presented is as important for the listener as is the method of speech production. Traditional methods of testing speech communication systems such as telephones or radio have relied extensively on articulation tests. In order to strip speech of the important contextual clues that enable a listener to infer what was said even though he did not hear it properly, articulation testing consists of using isolated words spoken in an arbitrary, linguistically non-significant order. This approach falls a long way short of assessing the usefulness of the communcation channel for genuine human conversation. So it is with computer-generated speech also. A system that can produce individually highly intelligible words, for example some random-access method of wave-form storage, will not necessarily produce easily understood speech when the individual words are assembled to make a sentence. Context plays an important role in speech perception: we are all familiar with the anticipation of football results from the way the newsreader reads them, giving different emphasis to teams with significant results, like a scoring draw.

In English, emphasis is signalled largely by changes in the prosodic features of speech, namely rhythm (rate of speaking) and intonation (voice pitch). Moreover, these changes are used to signal the meaning of a sentence. Thus a rising pitch at the end of a sentence often denotes a question. Consequently the prosodic aspects of spoken messages from computers should conform to normal human usage, otherwise there is the possibility that the listener may infer the wrong meaning.

It could be argued that it will be some time before we need true conversational output from computers, as they largely contain information of a highly structured and often numerical nature and consequently control over prosodic aspects is not important. However, a series of comparative experiments at Bell Laboratories[2] showed that people's ability to write down correctly a string of computer-generated telephone numbers was significantly improved if the correct rhythm and intonation were applied. Telephone numbers that were made from randomised recordings of isolated spoken digits were judged by the listeners to sound more natural but produced more transcription errors.

This raises the question of what the voice should sound like. There are several aspects to describing speech quality and these can be factored into two main headings: those that contribute to the understanding of the message and those that do not. In commercial systems the prime requirement is for good quality speech that is easy to listen to and does not cause fatigue on the part of the listener. This means that it must be clear and intelligible and possess the correct prosodic features that

facilitate understanding. The non-message related factors include such things as the fidelity of the voice – does it sound as though it was produced by a real vocal tract as opposed to an artificial one? – and the assumed personality of the character behind the voice. This arises from the fact that there is much more information conveyed by the voice than the text of the message being transmitted: for example, mood, character, dialect and so on. Whilst the message-related factors determine the understanding of the message, the non-message related factors are likely to determine human reaction. There is much to be said for providing a computer voice which is understandable but which possesses a distinctly non-human characteristic to remind the listener that it is *not* another person speaking. The balance of importance between the message-related and non-message related factors may well depend on the application, yet again indicating the need for a flexible voice output technique.

Another relevant aspect is the way that words are used to form messages. On a VDU screen the interpretation of a string of numerical characters is determined by the context or the position of the characters in a table of values. Thus 1933 could be a telephone number, a quantity or a year (or even a rather aged line-printer) and there might be no distinction between these representations on the screen. For this to be readily assimilated in a spoken form, however, the style of presentation should match the nature of the number, thus:

telephone number	one nine three three
quantity	one thousand nine hundred and thirty three
date	nineteen thirty three.

A speech sub-system which provides control over the prosodic aspects goes a long way towards providing a sufficient degree of flexibility to generate different numerical forms easily.

Other timing considerations have an effect on the listener and imply constraints in the way that any speech subsystem is controlled. Once started, speech output must be continuous with pauses only at semantically significant points like phrase and sentence boundaries. These pauses probably help the listener get the messages into and out of his short-term memory and, as every actor knows, the timing of these pauses has to be accurately controlled. The insertion of pauses at random is likely to have a disturbing effect on the listener and too long a pause may mislead him into thinking that the speech output has finished.

Another timing consideration is the speed of response to a listener's reaction. Because of the transitory nature of speech, together with the limitations of the human short-term memory, the listener must have the ability to cause the system to repeat the last message. As the use of speech is likely to be in interactive communication, delays of several seconds which are common and are acceptable in VDU-based systems will be quite unacceptable; the response to requests like 'repeat' or 'next please' must be short, of the order of 0·5 seconds. The implication of all these considerations and constraints is that computer output of speech must be controlled by an autonomous subsystem and not by a general time-sharing operating system which has to satisfy many competing demands for processing power.

3.2 *Programmer's requirements*

The listener's requirement for spoken output that is easily understood and remembered implies a controllable means of speech output. The programmer's task is to control this from the information and data available to him. The detailed control of flexible speech output is complex and requires a good knowledge of speech science and related topics. The programmer should be shielded from the details of how rhythm and intonation are controlled, but he should have available the means to control them. This principle can be extended to providing a means of specifying different types of numerical output and leaving the speech subsystem to work out how they should be done. The kind of interface envisaged therefore is similar to and an extension of the kind provided for the control of VDUs, where the programmer has the options of choosing lower case, italics, colour, flashing characters and so on. If the programmer is provided with an easy-to-use interface it is much more likely that he will be able to play his part in satisfying the listener's requirements.

The ideal in speech production would be the ability of the system to say anything that could be printed, which would imply the ability to generate speech from text. Techniques for doing this are being developed[3] but until they have matured to the level of commercial acceptability, any speech subsystem is going to be working with a restricted vocabulary of words and phrases. If the subsystem is to be responsive to changes in a user's business which lead to changes in the vocabulary, for example the introduction of a new product with a new proprietary name, there must be provision for changing the vocabulary easily.

3.3 *Manufacturer's requirements*

The equipment manufacturer has some requirements to satisfy in addition to the commercial ones of meeting his customers' requirements in the most cost-effective manner. One of the most important is the connectability of the equipment. Ideally it needs to be connectable to existing systems with the minimum amount of disruption, preferably using an existing channel.

The requirement to be able to supply new vocabularies to meet changing customer requirements means that vocabulary up-dates should be supplied on conventional computer media so that they can be loaded like new software. Moreover, a support service has to be provided with suitably trained staff that can carry out this vocabulary preparation. An important requirement here is that if the vocabulary is derived from real human speech, then access to the original speaker must be maintained and facilities provided to match new utterances to ones that have been provided some time previously: it is well known that speakers' voice characteristics change with the passage of time.

Although initial systems will use data derived from real speech, the long term requirement of synthesis from text, to handle name and address files for example, must be considered. Thus it is important for the manufacturer to choose a speech output technique that can be enhanced to meet this future requirement.

4 RADC speech output project

4.1 Speech generation technique: general considerations

In summary, a good speech output system must have the following properties:

(i) it must be capable of producing intelligible speech
(ii) the rhythm and intonation must be under programmer control
(iii) the storage requirements for the vocabulary must be reasonable
(iv) the technique used for the speech generation must be capable of enhancement to meet later expanded needs such as synthesis from text.

The simplest of all methods of speech generation is the use of prerecorded words and phrases which can be called in the appropriate order from storage. This has been used for small-vocabulary applications quite successfully but it does not meet the above requirements very well. The rhythm and intonation cannot be controlled without repeating the items in different forms; the storage demands are large, a minimum of 20 000 bits being needed for one second of speech; and the technique cannot be expanded to meet the needs for large vocabularies like names and addresses.

A satisfactory technique should use some method of synthesising spoken words and phrases from more basic material and should be modelled on the human speech production process. The best method seems to be that of *formant synthesis*[4]; this is the one chosen for the RADC project and will now be described.

4.2 Formant synthesis

The formant description of speech models the acoustic behaviour of the vocal tract in terms that are related to what the speaker does with his vocal apparatus when speaking, rather than to capture the detailed wave form of the spoken sound. The primary energy in speech comes from the flow of air from the lungs. In voiced speech (e.g. vowel sounds) the flow of air through the vocal cords causes them to vibrate at a rate determined by the air flow and the muscular tension applied to the cords. The resulting puffs of air excite the resonances of the air in the vocal tract, the frequencies of which are determined by the positions of the articulators, i.e. tongue, lips, jaw. In unvoiced speech (e.g. s, sh) the flow of air through a narrow gap in the vocal tract produces a noise-like sound whose frequency content is determined by the position of the articulators. The formant description of speech is a way of describing the frequency spectra of speech sounds in terms of the positions and amplitudes of the dominant peaks in the spectrum (Fig. 1). As the articulators move from one position to another, the formant peaks move accordingly, but because the articulators move much more slowly than the vibrating air molecules, a formant description of speech is much more compact than that of the acoustic waveform. A formant synthesiser takes as its input a formant description of speech (Figs. 2 and 3) and computes the values of a waveform according to the mathematics of the model incorporated within it.

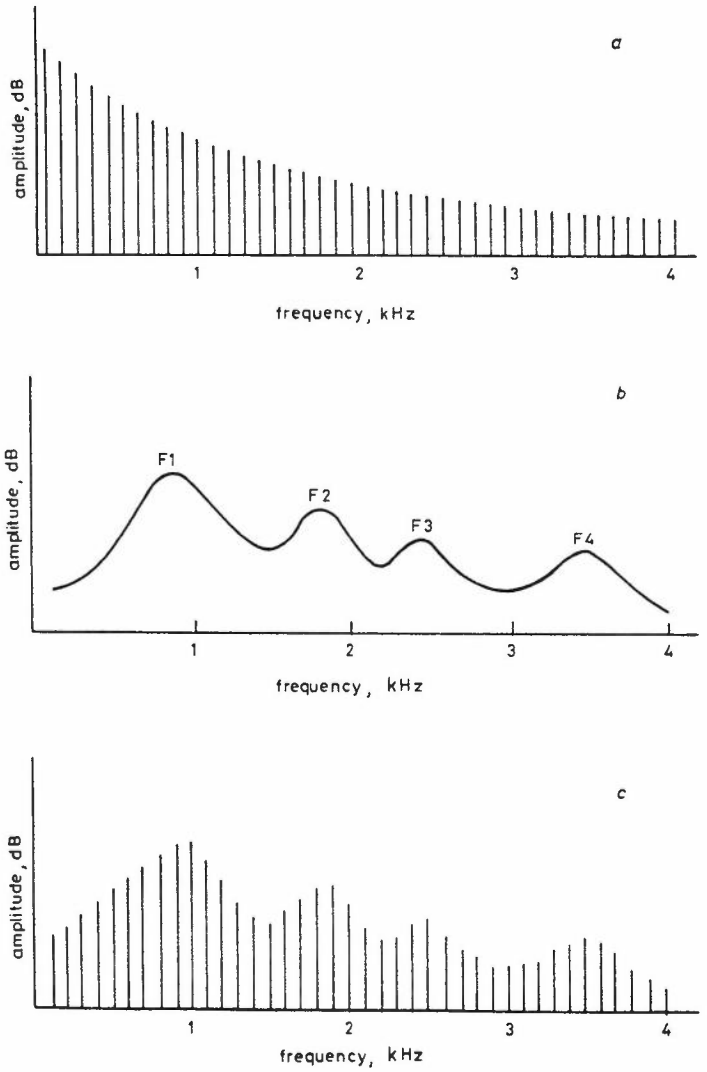


Fig 1 The formant description of speech describes the transfer function of the vocal tract in terms of the positions of the peaks in its frequency spectrum. The positions of these peaks depend upon the positions of the articulators (tongue, lips, etc.). The spectrum of a speech sound is the product of the spectrum of the source and the transfer function.

a Frequency spectrum of vocal cords during voiced speech
b Transfer function of the vocal tract for a typical voiced sound
c Frequency spectrum of the resultant speech sound

Formant synthesisers have been studied and used extensively by speech scientists.[5] Typically four formants (the four lowest resonant frequencies) have been found to be sufficient to produce highly intelligible speech. Three formants are sufficient for

telephone use with its restricted bandwidth. The formant description of speech is only an approximation to what happens in real speech, but if sufficient care is taken in deriving the formant parameters from real speech it is almost impossible to distinguish resynthesised speech from the real speech from which it was derived.[6] This more than meets the need to produce highly intelligible speech. Moreover the formant description of speech is a very compact one and good quality synthesised speech can be produced from a data rate as low as 1000 bits/second.

Time (ms)	ST	F0	F1	F2	F3	A1	A2	A3	F	AF
10	0	140		0	0	6	-48	-48	0	-48
20		138	234			25				
30		125				28				
40		119	156	1638	2691	31	-30	-30		
50		103	195	1599	2652					
60		102	234	1560	2730	34	-42	-24		
70					2769	37	-24			
80		101		1521			-30			
90				1482						
100										
110		100	546	1638	2808	40	-12	-12		
120		101	585				- 6			
130		102	624	1560		43	-12			
140		103		1521			- 6	-18		
150		104								
160			663	1404						
170			702	1326	2847					
180										
190				1287	2886			-24		
200										
210										
220		105								
230										
240		104								
250		103		1326			-12			
260		100	663	1404	2647					
270				1521		40	- 6	-18		
280		99								
290		100		1560	2808					
300			624	1638	2769					
310				1716			-12			
320		101	546	1872	2691	37	- 6	-12		
330				1950			-12	-18		
340			507	2028						
350				2067			-18			
360			468	2184	2652	34				
370			429	2223				-24		
380		100	390	2262			-24			

Fig. 2 continued on facing page

Time (ms)	ST	F0	F1	F2	F3	A1	A2	A3	F	AF
390		101			2691					
400		102		2301			-18	-30		
410		103	351	2106			-24			
420		104								
430		105	312	1443	2652		-30			
440										
450				1521	2769		-36	-24		
460		106		1482		31	-24	-18		
470										
480				1443				-24		
490										
500		120	0	0	0	6	-48	-48		

Fig. 2 Synthesiser parameters for the word 'NINE'
Each line represents 10 ms of speech. Blanks indicate that the parameter value remains unchanged
ST: Sound type 0 = voiced, 2 = voiced fricative, 3 = unvoiced
F0: pitch of the voice in Hz
F1, F2, F3: first three formant frequencies in Hz
A1: overall amplitude in dB
A2, A3: relative amplitudes of F2, F3 in dB
F: unvoiced sound type
AF: relative amplitude of unvoiced sound.

The use of formant synthesis also meets the requirement of being able to control prosodic aspects. The parameters that are used to control a formant synthesiser (Fig. 2) can be manipulated independently of one another, and these include the voice pitch. Thus in order to modify the pitch of the synthesised voice it is necessary to change only the values corresponding to this parameter. The speed of talking can be varied by altering the rate at which parameters are fed to the synthesiser. A typical up-date rate of parameters is 100 times a second. If the same parameters are sent at twice that rate, the synthesiser model is driven more quickly and the resulting speech is twice as fast but with none of the 'Donald Duck' kind of distortion associated with speeding up a tape recorder. In practice, the control of speaking rate is more complex than the simple overall alteration of the rate at which parameters are supplied to the synthesiser. The perception of some speech sounds is influenced strongly by dynamic effects, that is by how rapidly the articulators move from one position to another, whilst the perception of steady-state sounds like vowels is largely independent of changes in the rate of speaking. The need to control carefully the way speed changes are carried out is a very good reason for shielding the programmer from this kind of complexity.

It is not only the synthesiser design that determines the quality of the speech produced, but also the values of the parameters that drive it. There are three main sources of data from which these values can be derived. The first is by analysis of real speech. This has been studied extensively and has been found to be difficult to

automate completely.[7] With careful manual intervention and editing, excellent quality speech can be produced but it is an expensive process. The RADC approach, to be described later, is to employ a combination of computer and manual methods and has proved more cost-effective.

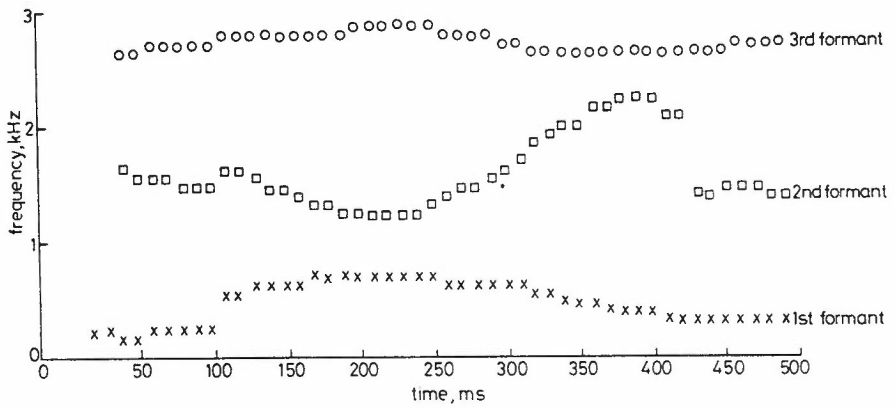


Fig. 3 Formant representation for the word 'NINE'

Another method is what is called synthesis-by-rule.[8] This is based on the linguistic premise that a spoken utterance can be described in terms of discrete sounds, in the same way that a written message can be constructed from the letters of the alphabet. Every speech sound can be described in terms of a set of synthesiser parameters, together with rules that determine how the parameters are modified by the context of the preceding and succeeding sounds: the dynamics of the articulators are such that these cannot be moved instantaneously from one position to another. The input to a synthesis-by-rule system is a string of phonetic elements specifying what has to be said, together with some parameters that specify rhythm and intonation. Thus synthesis-by-rule is a completely generative method, but the present state of the art is such that it requires trained and experienced operators to produce good quality speech.

The third method is synthesis from text. Synthesis-by-rule enables the programmer to specify any speech output but requires control at the phonetic level. Synthesis from text takes the method a stage further by providing a mechanism for converting the orthographic representation into a phonetic one. The difficulty here is that English in particular is a mix of rules and exceptions – a classic example is '- ough', for which there are several pronunciations depending on the context. Synthesis from text will provide the ultimate means of speech output, enabling material to be printed or spoken from essentially the same data. But because of the vagaries of pronunciation it is unlikely that any automatic system will be capable of producing the correct one for all names and addresses, for example. It is likely that an intelligent speech subsystem would deal automatically with the common names but that

some additional pronunciation guide would be included with other items. This would be phonetic in character and would occupy no more than an amount of file space equivalent to that of the orthographic representation. At present, synthesis from text has not matured sufficiently to be commercially acceptable or economically viable, significant amounts of computation being needed to convert from text to speech, along with the rules necessary to impart the correct emphasis.

The requirement for commercial systems is not to be able to read English prose but to speak aloud information of a much more restricted kind and content, such as 'the customer's name is . . . and the address is . . . '.

Until the completely generative approaches of speech synthesis have developed to the point of commercial quality and viability, speech output based upon formant synthesis will use parameters derived from real speech, giving good quality speech at modest storage costs with the desired degree of control and flexibility. When the time comes to use the generative approaches it will still be possible to use formant synthesis, but controlled in a different manner. Thus the use of formant synthesis provides an open-ended development path for customer and supplier alike.

4.3 Hardware implementation

Analogue techniques were used in the early synthesisers.[8] Bandpass filters whose centre frequency could be controlled were used to simulate the formant resonances and these were excited by either a controllable periodic source or a nosie source, corresponding to voiced and unvoiced speech, respectively. Analogue circuitry made from discrete elements is prone to drift and needs careful setting up and subsequent adjustment. The RADC decision, taken in the early 1970s, was to build a synthesiser using wholly digital techniques. At that time an economical design could be produced only by using a multiplexed technique whereby the synthesiser could be shared among a number of channels, because of the stringent need to compute a new wave-form sample every 100 μs. A design was produced that enabled 32 independent output channels to be supported simultaneously.[9] Subsequent developments in semiconductor technology have made it possible to build a synthesiser using a standard microprocessor and this has formed the basis of the current implementation. It is now possible to put a formant or other synthesiser on a single chip and several semiconductor manufacturers have developed such chip sets for mass-market applications including electronic toys. However it is the control aspects that are important for the commercial market and consideration of these was a leading factor in the design of the RADC system.

Fig. 4 shows the prototype synthesiser controller which has been developed to provide the degree of flexibility and control required in a high-quality system. The aim has been to design modular low-cost equipment that could be readily attached to existing computer systems and easily enhanced to provide a multichannel capability. The three main components are an 8-bit microprocessor with store and I/O ports, a synthesiser and Touch-Tone signalling detector and a line unit to interface to telephone lines.

The controller uses an 8-bit industry-standard microprocessor with 64 kbytes of RAM storage and serial and parallel interface ports. The serial port is used

for the mainframe link and the parallel ports for controlling the synthesisers, signalling detectors and line units. Although two independent speaking channels are shown, only one has been implemented in the prototype. The storage is used for both code and vocabulary data, there being room in the prototype for approximately 100 seconds of speech, or 200 words of half-a-second duration. As this is considered to be adequate for many applications, no particular effort has been made to produce a more compact representation of the data although there is scope for reducing the data rate further.

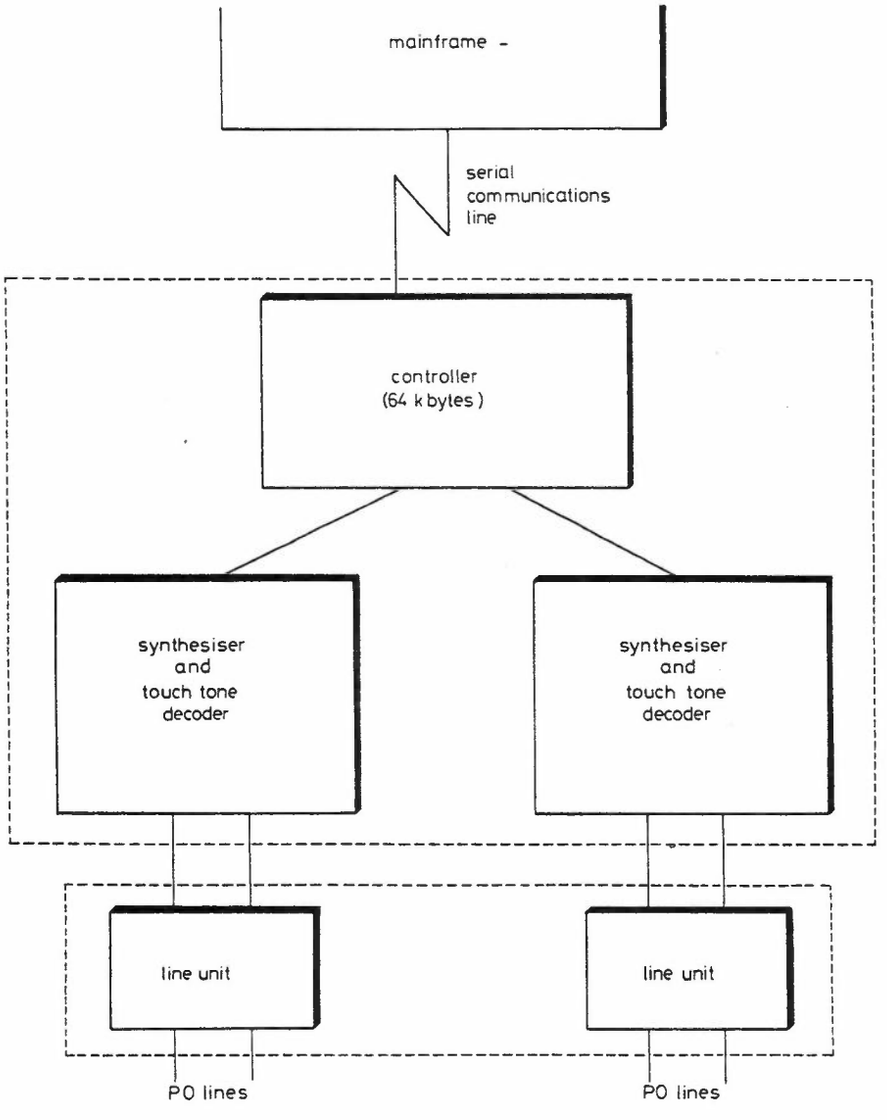


Fig. 4 Block diagram showing the main hardware units of the prototype speech output sub-system

The synthesiser uses another microprocessor with associated support circuitry in an implementation of the technique used by Underwood and Martin.[9] In many applications it will be necessary for the end-user to have some means of communication with the speech subsystem. In the long term, speech recognition techniques[10] will have matured sufficiently to make two-way speech communication possible. Until then the input will be by means of a limited function keyboard, similar in concept and lay-out to that used with Prestel. However, the signalling method normally used in voice response systems (MF4) employs the selective use of two tones out of eight, as used in Touch-Tone telephone dialling in America. Until that signalling method or some other becomes an integral part of every telephone, the keypad will be a separate portable unit that acoustically couples to the microphone of a standard telephone handset. The synthesiser board incorporates circuitry to decode the pairs of tones that are transmitted down the line by such a terminal.

The line units provide the necessary interfacing protection between the speech subsystem and the telephone network. The unit has provision for both receiving and making telephone calls under program control.

4.4 Synthesiser control

The code in the controller performs three types of function:

(i) detailed control of all the I/O: maintaining data flow to the synthesiser, receiving data from the Touch-Tone detector, controlling the line unit and servicing the communications protocol
(ii) provides the high-level interface for control of speech output
(iii) provides the interaction management, independent of the mainframe.

The most interesting of these is concerned with the control of the speech output. Given that the synthesiser approach enables the style of output to be controlled, the question arises as to how this should be done. It is envisaged that different levels of control are appropriate to different application, so a number of levels have been implemented.

At the lowest level, each vocabulary item can be called by item number; these items can be of any length and any one can be a phrase, a word or a part word. This is a most flexible arrangement and enables the best compromise to be made between naturalness and efficient use of the store. If all the words in a particular phrase appear nowhere else in the vocabulary, the most natural utterance results from storing it as a complete phrase. If a word or part word occurs in several contexts, more efficient use can be made of memory by storing the item once only and concatenating it with different vocabulary elements to form complete words.

It is likely that when a vocabulary item is used in different contexts it needs to be emphasised differently. For example, the 'six' in the word 'six' is shorter than the 'six' in 'sixteen'. To accommodate changes in emphasis two optional control characters can be used to control the pitch and speed of each item independently. As noted earlier, changes in the rate of speaking cannot be applied uniformly to the

speech, so the vocabulary data contain information that allows speed changes to be carried out in an appropriate manner.

Whilst this level of control provides the programmer with a powerful way of changing the speech and at the same time shelters him from the details of how it is done, there is a need for some commonly-used strings of vocabulary items to be handled more easily. One example is the different forms of numerical output which are likely to form a part of most application vocabularies. To save the programmer from having to remember and to use explicitly the rules for generating quantities – the machine should say 'thirty three', not 'threety three' – these rules are coded in the controller and enable the different types of output to be generated automatically. Thus:

N(digit string)	causes the digits to be output with suitable pauses and prosodic changes to make them easily understood
O(digit string)	gives the ordinal form: first, second . . twenty-fifth . .
Q(digit string)	causes the digit to be spoken as a quantity e.g. one hundred and seventy two

Another feature is the message facility, whereby a string of user-defined vocabulary items can be referred to by a message number. Moreover, this facility is parametrised so that variable information can be included.

A particularly important requirement for a speaking system is that the listener must be able to cause the last message to be repeated. It is a natural human reaction, when asked to repeat something, for the speaker to say it again more slowly and deliberately, so that the listener will have a better chance of understanding it the second time. The controller is provided with a facility which mimics this aspect of human behaviour.

The controller also provides a means for requesting and handling Touch-Tone data and for controlling the line unit. In order that the controller can react quickly to events such as the input of data or detection of ringing current on the line it must do so autonomously without reference to the mainframe. The prototype enables sequences of instructions, using all the facilities which it controls, to be stored and actioned. This enables the listener and the controller to interact independently of the mainframe until a point is reached at which mainframe action is required, like the retrieval of some stored information. This is analogous to the filling in of a form on a VDU screen, which goes on under local control until the 'send' key is depressed.

4.5 Vocabulary preparation

Until the truly generative approaches to speech synthesis are mature enough to produce the high quality speech required for commercial systems, thereby enabling the computer user to develop his own vocabularies, vocabulary preparation will be a specialised bureau-type service that will have to be provided by the supplier of the

speech subsystem. Vocabulary preparation is the crucial factor in using a formant synthesiser, as it largely determines the quality of the synthesised speech. The major requirements here are:

(i) it must be capable of producing good quality speech;
(ii) the service must not be costly to operate, either in terms of the equipment it uses or the time it takes;
(iii) the level of skill required to operate it must not be great;
(iv) it should contain facilities to enable vocabulary up-dates to be closely matched to earlier vocabulary items as a customer's requirements change.

The analysis of real speech to provide parameters for a formant synthesiser is a complex process and is only one stage in preparing a vocabulary for a particular application. The major stages are as follows.

The first stage is to list and analyse the utterances that have to be made and to prepare a script for recording purposes. Although the formant synthesis approach enables synthetic speech to be massaged to suit the context, experience in vocabulary preparation shows that the best results are produced if the words are recorded in a context which is identical or as similar as possible to the context in which they are to be used. The next stage is to produce a studio-quality recording of the script, using a speaker with good voice control. Information that is lost at this stage cannot be recovered, so it is worth taking care with this part of the process.

Once the recording has been checked for correct pronunciation, signal level etc., it is transferred to a conventional disc file via an analogue-to-digital converter. The resulting digital recording is listened to and then split up for subsequent processing. A suite of analysis programs has been developed at RADC to produce files of synthsiser parameters, which then require manual editing. The editing process uses an interactive program that enables the parameters to be modified and the resulting synthetic speech to be listened to. The need for this editing arises because the analysis programs do not handle all speech sounds equally well, so that mistakes have to be corrected and omissions repaired. More importantly, such processes as defining the boundaries of vocabulary items and adjusting the balance (loudness, pitch etc.) between them require human skill and judgement. The balance of man-time and machine-time in vocabulary preparation is such, however, that further automation of the process in its current form would not be cost-effective.

The final stage of the vocabulary preparation is the encoding of the synthesiser data into its compact form and the preparation of files suitable for loading into the synthesiser controller.

4.6 Current status

The project has reached the stage where the design principles of all the required software and hardware components have been verified with the prototype implementation. A number of trial applications have been modelled using simulated data-

bases. The next stage is to mount realistic field trials in order to gain first-hand experience in applying the techniques to practical computing tasks.

5 Applications

The kinds of situation in which speech output could be valuable have been indicated earlier in the paper and are summarised in Table 1. Several systems have been produced commercially, particularly in the USA, within the following fields of application:

order entry by mobile salesmen
status reporting, for example for work in progress
banking enquiries and transactions
credit authorisation
spares location
ticket enquiry and reservation
direct mail order.

Most of these systems have been remote data entry or database enquiry, where voice has been the sole means of communication with the user. So far, speech output has not been integrated into existing terminal facilities. Once this happens there are likely to be many instances where the use of voice provides a valuable adjunct to those facilities. Some initial applications might be to the following tasks:

(i) to provide verbal feedback of data that has just been entered, allowing complementary use of eyes and ears for checking;
(ii) to provide instructions to operators in large computer installations;
(iii) where a screen is already full of information and a simple message is required to guide the user, for example, Computer Aided Design (CAD) or Computer Aided Instrumentation (CAI)

Table 1. Situations where speech output could be valuable

• The end-user is mobile, so that any telephone could become a terminal using a highly-portable acoustically-coupled keypad.
• There is a requirement for a large number of widely distributed users, particularly if the access by any one user is relatively infrequent and does not justify the use of an expensive terminal.
• The volume of data to be input or output as part of any transaction is small and relatively simple in nature.
• The information to be disseminated is constantly changing (hard copy would be out of date).
• There is a need for a database to reflect an up-to-the-minute situation. This includes order entry systems for products with a short shelf-life, as well as stock control systems.
• A service has to be provided on a round-the-clock basis.
• In announcements or alarms where the attention of one or more individuals must be drawn to a message.
• Where a message needs to be communicated privately to a user.

It is inconceivable that such an important and powerful means of communication as speech should not play a significant role in the man–computer communication of the future. Ultimately speech output and input are likely to be an integral part of every computer, just as VDUs are today. However, from our current viewpoint where we are only just becoming accustomed to the speech technology that is now available it is very difficult to predict just how widespread its use is going to be.

Two main factors are likely to affect the use of speech in the long term. The first is what computers will have to talk about. Developments such as CAFS[11] mean that it is now possible to design information systems that are able to answer requests quickly enough to make conversational working a real possibility. Speech output needs the change in emphasis from data processing (computer-oriented facts) towards information processing (human-oriented facts) that is now taking place.

The second factor is the human reaction to the idea of computers that talk. Consider the likely effect on a computer operator for example when the computer booms out in an authoritative voice 'load tape three six seven five'. Such a message may well be acceptable when displayed on a screen, but the wrong tone of voice from the computer may have an effect which is entirely contrary to the user-friendly one we are trying to create. Although we cannot predict what the human reactions to computer speech will be, we are now in a position to use the technology in real-life situations in order to assess them. The next step forward is to start to apply speech output to practical use.

6. Conclusion

This paper has attempted to show that there are a number of aspects other than the purely technological ones that have to be considered when designing equipment to give a computer a voice. Good design is a compromise between ends and means, goals and constraints.[12] When designing machines that will use speech, with all its linguistic, cultural and psychological implications, it is clear that consideration of human requirements must play an important role. Requirements can often be met without regard for cost; but the solution described in this paper not only satisfies these human requirements but does so in a highly flexible, modular and cost-effective manner that provides a development path for the future.

Acknowledgments

The work described in this paper is the result of a team effort in RADC over a number of years and the author acknowledges with gratitude the contributions of all those who have participated. Particular mention should be made of M.J. Martin who has been responsible for the design and implementation of the equipment, and of the late Roy Mitchell for his insight and guidance; and of Gordon Scarrott, Manager of RADC, for creating and sustaining the environment in which the work was done. Grateful acknowledgments are due also to the Department of Industry which, through the ACTP scheme, gave valuable support in the early stages of the project.

References

1 MILLER, G.A.: The magical number 7, plus or minus two. *Psychological Rev.*, 1956, **63**, 81-97.

2 RABINER, L.R., SCHAFER, R.W. and FLANAGAN, J.L.: Computer synthesis of speech by concatenation of formant-coded words, *Bell System Tech. J.* 1971, **50**, 1541-1588.

3 ALLEN, J.: Synthesis of speech from unrestricted text, *Proc. IEEE* 1976, **64** (4), 433-442.

4 FLANAGAN, J.L., COKER, C.H., RABINER, L.R., SCHAFER, R.W. and UMEDA, N. Synthetic voices for computers, *IEEE Spectrum* 1970, 22-45.

5 FLANAGAN, J.L. and RABINER, L.R. (Eds), *Speech synthesis*, Dowden, Hutchinson and Ross, 1973.

6 HOLMES, J.N.: The influence of glottal waveform on the naturalness of speech from a parallel formant synthesiser, *IEEE Trans. Audio & Electroacoust.* 1973, **21**, 298-305.

7 SCHROEDER, M.R.: Parameter estimation in speech: a lesson in unorthodoxy, *Proc. IEEE* 1970, **58** (5), 707-712.

8 HOLMES, J.N., MATTINGLEY, I.G. and SHEARNE, J.N.: Speech synthesis by rule, *Language & Speech* 1964, 7 (3), 127-143.

9 UNDERWOOD, M.J. and MARTIN, M.J.: A multi-channel formant synthesiser for computer voice response, *Proc. Inst. Acoust.* 1976, 2 - 19 - 1.

10 UNDERWOOD, M.J.: Machines that understand speech, *Radio & Electron. Eng.* 1977, **47** (8/9), 368-378.

11 MALLER, V.A.J.: The content-addressable file store CAFS, *ICL Tech. J.* 1979, **1**(3), 265-279.

12 SCARROT, G.G.: From computing slave to knowledgable servant, *Proc. R. Soc. London Ser. A.* 1979, **369**, 1-30.