17

Maintenance of Large Computer Systems – The Engineer's Assistant

M. H. J. Baylis
Atlas Computer Laboratory
Chilton, Berkshire[1]

SUMMARY

This paper describes some of the difficulties in maintaining large computer systems and considers how machine perception techniques can be applied to the problem. A program called the 'engineer's assistant' is described as a step in the right direction.

Elaborations are made on the meaning of 'the right direction', and an experimental implementation of some of these ideas is described.

INTRODUCTION

A large computer system is usually maintained by a team of engineers. Inevitably these tend to be of different abilities, and some of them become more knowledgeable and therefore more proficient than others in specific areas of the machine. Although computer 'down time' has become more expensive by an order of magnitude on present large systems compared to the last generation, the engineers are not mending faults an order of magnitude faster. Moreover some faults which can occur nowadays are considerably more complicated than last generation faults. Therefore the best engineers are usually called upon in times of trouble, and these increase their experience at the expense of the other engineers. When these expert engineers move on to other jobs they take with them an invaluable sum of accumulated knowledge about the machine.

Further, when a unit gives trouble after a lengthy period of successful working, everyone present has forgotten how to proceed in tracing the fault. Symptoms often show which cause engineers to say 'we had this trouble two years ago, what did we do then?'

As a better than typical example, consider the present maintenance system on the I.C.T. Atlas computer at S.R.C., Chilton, remembering that Atlas was

[1] Now with I.C.T. Ltd.

designed around 1959. The engineers have a number of programming aids for preventive maintenance and fault diagnosis.
These include

(i) one or more stand-alone test programs for each part of the machine;
(ii) an engineering operating system which will multi-process many of these test programs;
(iii) a test-programming system which can run tests as object programs during normal computing service;
(iv) the operating system itself.

During scheduled maintenance time, (i) and (ii) are used for preventive work on those parts which have not been inspected during computer service time with (iii).

The operating system includes a number of tests. Instead of 'idling' in those periods when it is impossible to carry on with object program execution, random tests are made on the arithmetic units, and every five minutes a few tests are made on special equipment such as the instruction counter and the digital clock.

For all incidents of machine failures, the operating system outputs what information it can about the state of the machine and the malfunction. This maintenance system works well for finding normal faults but is of very limited use in finding obscure faults.

In the last few years significant improvements have been made by computer manufacturers in maintenance methods. As the cost of computer components has dropped, it has become feasible to provide much more checking logic in the machine. This extra logic allows for quicker detection of malfunctions and frequently permits the current state of the machine to be preserved for inspection. However, it seems to me that these improvements are not of much help in finding obscure intermittent faults.

FAULTS AND THEIR FINDING

We might group faults into two broad classes: those which provide symptoms regularly (solid faults), and those which provide symptoms very intermittently.

The first class is, of course, the easier to investigate. Such aids as test programs, oscilloscopes, etc., are well understood. The second class provides much more interest. How does one find a fault which shows itself once in an hour or once in a week? The answer, one likes to think, is by logical deduction from the evidence, backed up by intuitive guesses as to what could go wrong, based upon knowledge of previous machine behaviour. At present rather haphazard techniques are followed, such as the following:

(i) the incident might be ignored, with the hope that it will either go away or develop into a more solid fault;
(ii) engineers' tests or tests made from user programs are run for lengthy periods while various engineering attitudes are struck (e.g. varying the voltage margins);

(iii) guidance is sought from possibly helpful system program writers as to what was going on in the machine at the time of trouble.

When a large machine is first commissioned, intermittent faults tend to be hidden in the noise level, compared to solid faults. After a year or so the ratio is more than 80:20 in favour of intermittent faults. One reason for this is that multiple machine failures occur for each intermittent fault before it can be eliminated.

The general solution to this problem of minimizing down time must be to automate the processes of deduction as far as possible. The solution presented here consists of some primitive hardware and a sophisticated computer program to supplement the standard maintenance methods. The name 'engineer's assistant' is given to this sytem. We consider the program first.

OBJECTS OF THE ENGINEER'S ASSISTANT PROGRAM

The major program objectives are, in order of complexity:

1. to accumulate a memory of the symptoms and remedies for all previous faults;
2. to make this information available as needed:
   (*a*) to aid engineers in fault-finding;
   (*b*) to produce statistical information about the machine;
3. to provide question and answer guides to aid fault-finding;
4. to investigate and diagnose faults with little or no human assistance.

Taking these objectives in order, it will be realized that the major problem in implementation of 1 and 2 is that of data structures. The program is required to hold information about many interrelated objects; to access, add, delete, modify, and do calculations on objects and their properties, including such properties as interrelationships with other objects.

The work done in implementing a primitive engineer's assistant program on Atlas has led to a belated recognition that the correct data structure is a 'plex'. Plexes, like hash tables, are something that everyone invents for himself, but the formal definition and exploitation is due to Douglas Ross of M.I.T. (*see* Ross, 1961, and Ross and Rodriguez, 1963).

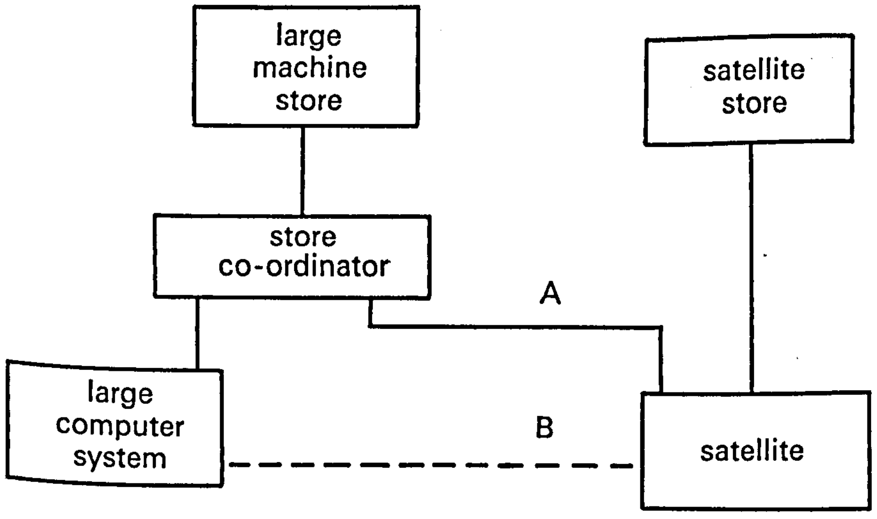
The name is taken from plexus–a network, or interwoven combination of parts in a structure. A plex is an interconnected set of *n*-component elements. The components of an element are its properties usually contained in successive addresses within the element. Typical entries would be the type and name of the element, and information which might range from binary bits, instructions, and symbolic data to different sorts of link connections with other elements. A moment's thought shows that most, if not all, data structures can be regarded as examples of simplified plexes. The tree structure of LISP (*see* LISP 1.5 programmer's manual, 1966) consists of elements with only two components; the 'ring-structure' of Sutherland's Sketchpad project is another example (*see* Sutherland, 1963).

For our particular application there needs to be a considerable number of types of element, e.g. installation, hardware units, sub-units, packages, faults, etc., where all elements of any one type have a consistent layout of information. The interlinking of elements will not only start off fairly complicatedly but will get more and more involved as the memory matures. For example, an element of type 'package' with name '890 board' might be reached through the route 'installation–(S.R.C. Chilton), area–(core store system), unit–(stack 5), fault–(digit 16 dropout)' etc. Or by the route 'package type- (read amplifiers),' or, again, by 'fault- (any), position–(MO3/10.4/16), date–(1/1/67 to 8/1/67) . . .', and so on.

So far, about thirty element types can be seen as necessary. Having decided on a data structure, the next decision is to choose a language in which the problem can be expressed. If most of the program can be generalized enough to be machine independent, then it will be that much more valuable. At M.I.T., extensions have been made to ALGOL by Ross *et al.* for handling plexes. There is some evidence that ALGOL X will be suitable. Another contender is CPL.

Providing a question and answer guide for fault-finding is relatively easy, assuming that the best engineers can explain how they mend faults. There is one essential requirement: a facility for the user to modify the questions and answers and provide new alternatives. In fact, for such a program to be really machine-independent it must start off asking for questions and enquiring what answers can be given. In order that the program can use its memory for finding any similar faults, the format for such a dialogue must be in the same form as the definitions given to elements in the plex.

We now come to consider the automatic investigation of faults. It should be emphasized again that, although the system will work for solid faults, it is the difficult intermittent faults that we are interested in finding. These may be of two types: either they cause a machine stoppage or they do not. In either case the fault may be in the central machine, that is, in the logic which controls the processing of instructions, and in practice it is these faults which are hardest to track down. We can envisage an engineer's assistant program (EAP) running in a partially broken machine, but because that may not be always possible it seems desirable that there should be a separate satellite computer for the program. The minimum connections needed are as shown in figure 1. With such an arrangement the EAP could follow its own question and answer route independently, that is, it could phrase questions and select some machine instructions which could answer them. This code would be positioned in the large machine store and the large computer instructed to obey it. Note that a body of code can be an element on a plex structure. There is little doubt that such a system would work. It would, however, have only limited success because, with many intermittent faults, the symptoms are transient and disappear before they can be recorded by any program. Also, symptoms manifested by programs are at present far too general; in the absence of more detailed evidence there can be too many possibilities to investigate. There is,



A: the satellite can access some or preferably all of the large machine store

B: a control link at interrupt or high level

Figure 1. Basic connections for maintenance satellite

therefore, a requirement for much more intimate evidence to be accessible by program, and without such evidence progress will be slow.

HARDWARE CONSIDERATIONS

Most computers have an engineer's console on which rows of lamps flicker. These lamps show the states of important control points in the central computer, and if the machine stops then the lamps give some idea of what was going on at the time.

Some observations are worth making:

1. There are not nearly enough lamps on most if not all computers. It should be easily possible to monitor all decision points in the logic.
2. The setting of lamps should be incidental to the more important function of setting bits in registers which can be accessed by program.
3. It should be possible to 'freeze' the states of these lamps and registers when specified conditions are satisfied, and to allow the computer to continue without altering these states. By having access to such powerful evidence the EAP run from a suitable computer should be able to make rapid and effective diagnoses. The freezing of states can be done in a number of ways.

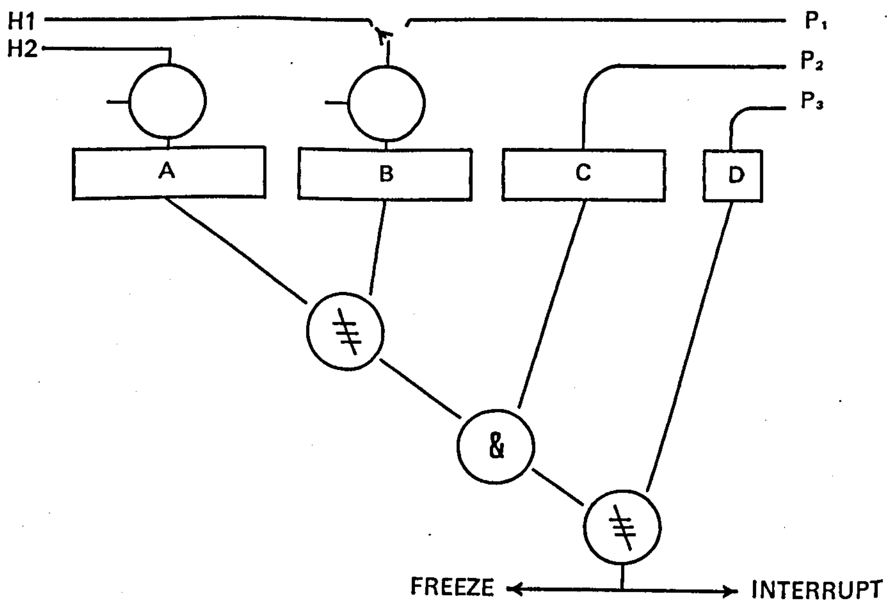


Figure 2. A possible simple check unit

Consider the check unit shown in figure 2. A, B, and C are buffer registers, probably equal in size to the machine word length. D is a one bit register. A and B can be set by hardware events H1, H2. B can alternatively be set by program P_1. C and D are set by program P_2, P_3. With this checker, A and B can be compared over a field C; D will then determine whether to freeze on equivalence or non-equivalence. If it is required to freeze when there is a difference between H1 and H2 in field C, then D will set to a zero, for example.

It is not feasible to position these checkers everywhere they might be wanted. Sets of floating plugs and sockets could be provided and the program or engineer could decide where they should be placed. A better method would be a switching matrix so that the EAP could connect the checkers it had to the parts of the machine in which it was currently interested. As a speculative matter, if checkers could be connected together under program control then very sophisticated conditions for freezing could be set up.

To our basic connections between a large computer system and a maintenance satellite we have now added special hardware to provide detailed information in the form of 'snapshots' for a perceptive program. We now leave these ideas and consider what has been implemented so far at the Science Research Council Atlas Computer Laboratory by S.R.C. and I.C.T. in collaboration.

THE ATLAS ENGINEER'S ASSISTANT

This program was mainly implemented by Andrew du Plessis (S.R.C., now Cambridge University) with the assistance of the chief maintenance engineer Peter Dean (I.C.T. Ltd.) and myself (*see* du Plessis, 1967). Its object was to explore the possibilities of a systematic data retrieval system applied to maintenance engineering. Most of the program consists of fault location procedures for the different parts of the machine. By running the test programs specified and thus answering the questions asked, some progress towards diagnosing faults will be made. When the program has reached the limit of the assistance it can give (it will then be pinpointing a faulty area, a particular group of packages, or an individual package), it can output records of all previous repaired faults which have followed the same diagnostic route. An underlying assumption here is that a component which has failed once is likely to fail again; in practice this is true enough on Atlas to make the program useful. It is possible to retrieve fault histories without going through the question/answer procedure, and to extract information about package types. The program is controlled on-line with a simple command language. After identifying itself, the program asks for a directive. Some of these directives are described below.

GUIDE 'UNIT'

This calls the fault location guide for the specified unit. Exit from the question/answer routine back to the command language is made by typing 'return'. The questions usually list alternative answers, and only the first few characters are necessary for valid replies. A typical example might be:

```
TYPE DIRECTIVE
guide main store
ARE ALL STACKS BROKEN? 'YES' OR 'NO'
no
IS FAILURE 'TOTAL' OR 'INTERMITTENT'?
total
IS TOTAL FAILURE EVIDENT IN 'LOSS OR PREPULSE', 'ADDRESS
TEST', OR ANY 'OTHER' TEST?
address
TYPE AS TEST RESULT 'SINGLE' DIGIT FAULT, 'BOTH HALVES' OR
'ONE HALF' FAILING
both
RUN ADDRESS TEST ON FAILING STACK. LOOK DOWN 970 AND
971 MONITORS TO SEE MOVEMENT. TYPE 970,971, 'STOPPED' OR
'MONITORS ALL MOVING'
monitors
CHECK ALL 972 CURRENT WAVEFORMS, THEN TYPE FOR PULSE
'O/K', 'HALF-SIZE' OR 'NONE'
half
```

```
PULL OUT VOLTAGE 964'S ONE BY ONE UNTIL PULSES GO FULL-
SIZE. REPLACE FAILING 964
IS A CASE HISTORY OF SIMILAR FAULTS REQUIRED?
yes
TYPE STACK NUMBER
11
THE CASE HISTORY WILL BE OUTPUT ON THE LINEPRINTER. IF
STILL U/S CHECK VOLTAGE LEVELS. TYPE DIRECTIVE
finish
```

BOARD *(b * Ua * p/q)*

The format for this directive is shown above where:

b is the particular package type number

Ua is a unit or sub-unit of the computer

p/q is a particular set of package positions in the machine.

The last part or last two parts may be omitted. Thus it is possible to obtain information about all packages of type *b*, to restrict this to a unit or to positions within a unit, e.g. the directive BOARD (822) would produce output of records of the form:

Installation Chilton
Serial number 131
Incident on 29/03/67
822 board, number 146309 in sub-unit R, failed at 169/4*16.12B6 causing fault type 03/B02.
PAR test 21 showed PAR 58 U/S. Board changed.

FAULT

The information retrieved by FAULT is a listing of faults which have occurred in a particular unit. The type of fault may also be specified. The format is similar to that for BOARD, and specifies the unit of the machine, an optional fault type and, optionally, a point in the fault location guide for that unit.

Other directives include DATE for setting start and end dates between which the program will operate, and PLACE. The latter commands information retrieval to be only from the specified installation. There are of course editing and updating programs for altering the memory. Engineers are forbidden to mend faults without contributing to the memory.

The system has been in use for some time. Its main uses have been in helping to train junior engineers and in presenting statistical information in a useful way. For example, it showed that a certain package position in the central machine had been changed significantly often. Once this had been observed, it was quickly realized that, although changing this package appeared to cure a fault, the trouble was in fact caused by an earlier logic stage.

Its limitations are that the data structure is not very sophisticated, the diagnosis tree cannot be altered easily, and the program is completely machine dependent. It has been observed that sometimes engineers cannot answer the questions. For example, the question may specify a monitor point and ask about a certain waveform; the engineer may not be certain what the waveform should look like. If the dialogue were conducted from a visual display unit instead of a typewriter, then the expected pulse shape could be displayed.

ATLAS HARDWARE MODIFICATIONS

Frank Fennel of I.C.T. Ltd. is modifying some of the S.R.C. Atlas central machine hardware along the general lines indicated previously. A second console has been built which monitors nearly every significant point in the machine. At present the monitored positions light lamps. Various freeze facilities are being added by means of simple checkers (the basic one consists of two floating buffers and a non-equivalence circuit). Plug and socket connections have been wired on the main frames so the checkers can be manually put into position. An assessment of engineering success will be made over the next few months which will decide how much further effort is put into the project. It has been found that to check some parts of the machine (e.g. function decoding) it is necessary to build checkers which duplicate the relevant machine hardware.

CONCLUSIONS

The system outlined in this paper might be of considerable use in the maintenance of large computer systems. A perceptive program which can obtain detailed information about the machine it has to maintain is not an unrealistic proposition. It is possible to envisage a satellite computer deciding to vary the voltage levels over a small area of a large machine, to monitor the behaviour, and to change packages without engineering intervention.

The situation will not be radically altered in the next generation of multilayer integrated circuit machines. For as far ahead as can be seen, if an area of a computer has intermittent failures there will be some interest in finding out about them.

There will be a need for a formalized accumulative record of machine faults both for local maintenance help and for deciding whether any failures are general enough to warrant package modifications or design changes.

The best test programs at present for a large computer are the operating system and user's programs. Engineers' test programs should eventually become redundant except for finding the simplest of solid faults–in which environment they are reasonably successful.

At the S.R.C. Atlas Laboratory a satellite will shortly be attached to Atlas. The machines will share a common disc file and also be linked to look like peripherals to each other (*see* Baylis, 1966, 1966a). This will provide a good framework, in view of the work already done, for a project to produce a more sophisticated 'engineer's assistant'.

REFERENCES

Baylis, M.H.J. (1966), Time-sharing on Atlas. *S.R.C. Atlas Laboratory publication.* Chilton: Atlas Computer Laboratory.

Baylis, M.H.J. (1966a), Satellite Survey 1966. *S.R.C. Atlas Laboratory publication.* Chilton: Atlas Computer Laboratory.

LISP 1.5 programmer's manual (1966), Cambridge, Mass.: MIT Press.

du Plessis, A.A. (1967), The Atlas engineer's assistant. *S.R.C. Atlas Laboratory publication.* Chilton: Atlas Computer Laboratory.

Ross, D.T. (1961), A generalised technique for symbol manipulation and numerical calculation. *Communs Ass. comput. Mach.*, 4, 147–50.

Ross, D.T. & Rodriguez, J.E. (1963), Theoretical foundations for the computer aided design system. *AFIPS*, 23, Spring J.C.C., 305–22.

Sutherland, I.E. (1963), Sketchpad, a man-machine communications system. *AFIPS* 23, Spring J.C.C., 329–46.