Use of high- and low-level languages in a real-time system

J. STENSON
Plessey Radar, The Plessey Co, England

Introduction

This paper discusses briefly some of the arguments for and against the use of high-level languages in a real-time system. Then it goes on to describe the results of efforts made by the Plessey Company to use a high-level language in parts of a real-time operating system.

It is hoped that our experience will be interesting to everyone who is looking at the use of high-level languages in real-time systems.

1. For and against the use of a high-level language

Consideration of whether to use a high-level language rather than a low-level one is influenced by the following factors:

1. Writing in a high-level language is much quicker.
2. Coding errors are less frequent.
3. Testing is often much quicker at the off-line stage.
4. Documentation is sometimes simpler.
5. Handover of the program is easier – whether between programmers or to the customer.
6. Even an efficient high-level language usually needs more space and more run time than a lower-level one.
7. If obscure bugs are found on line, the programmer will probably have to look at the machine code instructions to trace them, and this is easier from a low-level language than a high-level one.

A balance has to be reached between speed of program production, achieved by using a high-level language, and efficiency of code, achieved by using a low-level one. This is a very difficult decision because programmer time is always expensive, and any means of reducing it is worthwhile, but time and core space in most real-time systems are precious.

The solution may be to make high-level languages more efficient, so the object code they produce is nearly as good as a programmer can write. Or it may be to improve low-level languages to incorporate some of the features of high-level ones. Either or both of these improvements may be made.

When the decision between high- and low-level languages had to be taken we had time-scales to meet and limited space and time in the system. The rest of the paper shows how we decided between high and low languages, and the results of our choice.

2. Brief system description

This system is a multicomputer Air Traffic Control project. The computers are of two main types, and not all have access to the same facilities – for example, some have access to magnetic tape decks, some do not. All the computers can communicate with a common store in which the data base is held.

Each computer receives one interrupt at regular intervals, and from this interrupt all the system timing is derived, although there are other significant times, e.g. radar scan time and data link time, which have to be observed.

The system can be considered as having two parts, a 'foundation', which would be nearly the same whatever the system were used for, and 'application software', which is concerned only with the air traffic control functions of the system.

The 'foundation' consists of an operating system providing scheduling, communications, peripheral handling, reconfiguration, fault detection and location. Parts of this have been implemented during the past year, and it is from this implementation that the following information has been drawn.

3. The operating system

The operating system contains programs which perform the following functions:

1. Schedule tasks in every computer in system.
2. Provide communication between the computers in system and the common store.
3. Handle peripheral equipment – teleprinters,

punches and readers, magnetic tapes. Also the 'control panel', which is hardware device controlling the configuration of the system.
4. Control and information service to the operator. This program is known as The Director.
5. Detect faults.
6. Load programs.
7. Aid on-line testing.

4. Available languages

In this system we use two types of computer, the XL4, a double address machine, and the XL6, a single address, less powerful computer. For the XL4 we have a high-level language, MINICORAL, a subset of CORAL, and XAL, an assembler. For the XL6 we have XAL only.

MINICORAL can be a very efficient high-level language. An experienced programmer writing with store economy in mind can achieve a 1.1:1 size ratio – we have tried experiments and it can be done. But, of course, a less experienced programmer can produce much worse results than that; 2:1 is about average. There are very good off-line testing aids associated with the language, and this is a great advantage.

XAL is a mnemonic assembler with very good macro facilities. It is easy to learn, but there are fewer off-line testing aids associated with it, and therefore programs can take longer to debug. Really experienced XAL programmers have very little difficulty debugging their programs.

5. Available people

The programming team contained people with a wide range of programming skill. They ranged from those with 5 years' experience to those with none at all. Our trainee programmers were used mostly on the MINICORAL programs.

6. Split between high and low-level languages

When we looked at the list of tasks to be performed (Section 3, 1-6) some points were obvious immediately:
1. Some programs would have to exist in every computer in the system (scheduler, communication, on-line aids, fault detection).
2. Some were restricted to only a few computers because of the arrangements of the hardware (magnetic tape handling, control panel handling).
3. From the nature of their design some programs needed to appear in one (or only a few) computer only (The Director, Reload).

A simple split would be to use XAL for programs held in every computer, MINICORAL for those held in only a few. To a certain extent this was done but there were a number of weighting factors to be applied. These were:
4. Any program in the XL6 must be in assembler.
5. Some programs which appear in every computer in system have very low priority, and if their run time is a little bigger than it need be it may not matter very much, since the program will only run when there is spare time anyway.
6. Some programs which appear in every computer are run on rare occasions only in the operational system. Therefore their space and time need not be optimised to the same extent as others.

Starting from the position that we wanted to use MINICORAL if possible to reduce program production times we arrived at this split between MINICORAL and XAL:

MINICORAL	XAL
	Scheduler
	Communications
Control Panel Handling	Teleprinter Line
	Printer
	Paper Tape Handling
Magnetic Tape	Magnetic Tape
Handling (XL4)	Handling (XL6)
Director (XL4)	Director (XL6)
Fault Detection	
On-Line Aids	On-Line Aids
Reload using XL4	Reload from XL6

7. Review of each program

Each of these programs is considered below. The arguments for using the particular language are given, and where MINICORAL was used the results are considered. Where XAL was used less information is included.

There are two areas, Magnetic Tape Handling and The Director, where comparison between a MINICORAL and a XAL version of the same program can be made. It is not a direct comparison, because the MINICORAL versions were for XL4 computers and the XAL for XL6. The XL4 has more powerful instructions than an XL6, so one expects to find fewer instructions used in this computer.

7.1 Scheduler, communication

These programs were written in XAL because they are used in every computer in system, and therefore space is important. They are used con-

stantly in every cycle of work, and therefore time is important. We considered that any increase of space or time could not be allowed, and they were written by skilled assembler code programmers.

7.2 Teleprinter, line printer and paper tape handling

These were written in XAL because MINICORAL does not offer particularly good facilities for this type of peripheral handling, they were short programs anyway, and some were for the XL6. We did not spend very long over this decision – peripheral handling of this type seems to demand an assembler code.

7.3 Load

These parts of the load programs housed in an XL6 were written in assembler code, but where they were housed in an XL4 they were written in MINICORAL. This is because they are used comparatively rarely in the operational system (probably less than once every six hours) although they are heavily used during program development. It seems more important in a case like this to ensure that the program can be maintained easily and handed on from one programmer to another than to achieve minimum space or time. Space had to be considered, because there is limited space in the System. The programs did not occupy an excessive amount of core (0.7 K in 4 computers of 64 K capacity).

If we had to make this decision again we would make the same one. Any program used at a very low rate is a reasonable vehicle for an experiment in using high-level languages.

7.4 Control panel handling

The program resides in one computer only in the system, and is called every cycle. No significant amount of processing takes place unless the operator uses the System Control Panel to reconfigure the hardware in the system. This should be a rare occurrence – major reconfigurations involving computer changes should not occur more than once in 24 hours, and minor ones should happen less than once per hour.

In these circumstances no serious penalty in overheads will be paid at run time if the program is slightly slower than it need be. It was, therefore, written using the MINICORAL compiler.

The program was written by experienced programmers. It was completed quickly and with very few unforeseen difficulties. It operates well within the time available to it, and whenever the program has been run it has given satisfactory results.

This was another area where we feel we made the right decision. The ease of production and documentation justified the use of the high-level language, and no significant time penalty has been paid.

7.5 Fault detection

This program is scheduled at the end of the lowest priority list of tasks. It is called when all other work has been done, and having been completed adds itself to the bottom of the list again. Thus if there is very little work being done in the computer this program is run frequently, but as the work load builds up so the calls become fewer.

As the program will only be used when the workload allows it, the time overheads incurred if it is written in MINICORAL are not serious. The program resides in every computer in system, and therefore space is a matter of concern.

Despite the consideration of space the program was written in MINICORAL to reduce timescales. The space occupied is more than we allowed for when the program was designed, and we are faced now with a need to look through the program to find ways to reduce the space used.

Nevertheless, the program was ready on time and it can be used in its present overlarge state for a few months before the 'applications' programs increase in number and its size becomes a problem.

This is where the judgement between the two alternatives is difficult – we did achieve a result on time, which is important, but the excessive size of the program is just as relevant.

Summing up, it is apparent that this MINICORAL program is less successful than the two previous ones.

7.6 On-line aids

On-Line Aids will be used most frequently during the system development stage, at which run time and space overheads are not particularly significant. (It is possible to alter real-time situations just by using on-line aids, but this situation is unlikely to be made worse by longer run times.) The aids will be used occasionally in the operational system, but not often enough to make run-time overheads significant. Like 'fault detection' (Section 7.5), they exist in every computer in system, and space is important.

For reasons of speed of production, allied to a conviction that MINICORAL should be a good language for tasks like this, we decided to do two-thirds of the total work in MINICORAL.

The project suffered throughout its life from changing manpower, and was frequently understrength. Despite these difficulties it was completed on time: but its size is excessive. We attribute this to its low staffing level leaving no

time for a careful examination of the coding of each module to ensure that minimum space was used. Like 'fault detection', we now have to carry out this examination and reduce the size of the program.

It is possible that the same troubles would have been met if we had elected to write the program in XAL; in fact, it might have been even worse.

It is difficult to tell with this project whether it would have been better to write in a low-level language. It seemed to be a function of the low manning levels rather than an inherent difficulty in using a high-level language that caused the growth in size.

7.7 Magnetic tape handling

Only 2 of the XL4 computers are expected to use magnetic tape decks at any one time and they will not be in constant use. This was an obvious place to use MINICORAL and it has been used very successfully. At present the XL4 version is estimated at 2 K (with 75% completed, so the estimate should be reliable).

The XL6 version, written in XAL because there is no other choice, is 2.2 K.

These figures show a very close correlation between the MINICORAL and XAL versions.

7.8 Director

The Director has two parts, one of which resides in every computer in system, the other in one XL4 and XL6 only. The part that occurs in every computer in system was written in XAL for economy. The control part, in one XL4 and XL6 computer, runs once every cycle. This means that size and run time are not critical, although they cannot be ignored. The program for the XL4 was written in MINICORAL and that for the XL6 in XAL.

The sizes of these programs are: XL4, 2.2 K; XL6, 3.6 K. This is a very satisfactory result – XL4 programs should be smaller because the instruction set is more powerful.

8. Conclusion

In this multi-computer system it was possible to use a high-level language in those areas which were not common to every computer in the system. When a high-level language was used in an area which was common to every computer the results were less successful, although it is likely that the fault did not lie with the compiler.

From the experience gained in this system, designers may well be less sceptical of the wisdom of using high-level languages in real-time system than they have been in the past.

Acknowledgements

This paper is based on the work done by Plessey Programmers in System Implementation with the help and collaboration of members of the Royal Radar Establishment at Malvern.

Discussion

Q. You remarked that the most successful programs are those that are run least often. Is this because these programs are tested less thoroughly?

A. Not quite, but perhaps because they are less prone to interactions from other debugging, and errors in them have less far ranging consequences.

C. Good programmers want to get all the power of the machine, so they program in assembler. But the very best programmers want to get this in higher level languages.

C. Even when using low level languages we do not allow our programmers to use 'tricky code'. With high-level or macro languages one should accept some overheads and inefficiencies which result from enforcing programming standards.