The integrated control/distributed power software development shop

by JEAN-PAUL RENAULT
CIT-Alcatel
Velizy-Villacoublay, France

INTRODUCTION

In the constantly expanding spectrum of microprocessor applications, there is a large class of systems which used to be implemented with large or minicomputers, in which new software engineering problems have emerged. Typically, these systems involve quite a few hierarchical microprocessors—they perform fairly sophisticated control functions, and must comply with stringent reliability and availability demands. Though the distribution of functions significantly reduces the complexity of some of the technical problems usually encountered in such systems, the volume of software to be developed remains large, and system tests are still a problem.

Furthermore, the development systems offered on the market by microprocessor manufacturers or independent vendors are very well adapted to the development of small applications which involve a unique microprocessor and a limited volume of code, but they are insufficient for the development of large systems. The system builder engaged in distributed system development will, therefore, have problems to solve, until the ideal support software and hardware are made available on the market.

SOFTWARE DEVELOPMENT REQUIREMENTS

General requirements

What a programmer expects from a development system is now well understood for large and mini-computer software development, and since it has to do mainly with the way the user sees the system, or with the way he gains access to it, and not with the way it is built, there is no reason, no matter how different the target processor may be, that the same rules should not apply to microprocessor software development.

The qualities assumed of a development system have to do with ease of use and access, the overriding consideration being, therefore, that the system should be terminal-oriented. This should give the user unrestricted access to all system facilities from his desk. In addition, it should limit the need to handle media; cards should be completely abandoned, listings should not be used as the support of run results, but as reference only, when a program has reached a satisfactory level of completion, and magnetic media should be handled for archives, deliveries, etc., by the system itself, and not at all by the programmer. Even cassettes or diskettes, though handy, can create a great amount of confusion.

From his terminal, the user should have access to a certain number of programming aid facilities:

- Program handling aids such as text editors, library managers.
- Documentation support facilities, including document entry, document updating, document editing and printing, either directly on listings, or through phototypesetting for quality printing.
- Programming aids—Compilers, link editors, loaders, debugging facilities, etc.

Another extremely important feature of the development system is the command language it offers. It should be rich (i.e., it should offer many functions), and easy to use, which in fact means that it should be easy to invoke from a terminal, both directly through system commands and indirectly, through catalogued procedures built by the user to help him perform frequently repeated complex operations by means of simple commands.

Specific microprocessor requirements

Programming problems are not significantly different in distributed systems, even large ones, from those encountered in others. Programming proper is not a problem—high-level languages, when available, are perfectly suited for distributed system programming, and the fact that some specific microprocessor features are not accessible through them will never justify going back to assembly languages, except, maybe, in a few sections where performance is critical. This is true, in fact, of any system and of any high-level programming language. On the other hand, in some areas, programming will be even simpler. The kernels which operate the microprocessors are generally message-oriented,
and support only fairly static processes and connections between processes, which makes them easy to understand and develop. Even reconfiguration mechanisms, generally quite sophisticated in other systems, tend to be rather simple in a distributed environment, at least from the programming point of view.

The problems associated with unit testing are of the same order of magnitude as those traditionally found in systems—depending on the detailed internal architecture, it will be more or less practical and economical to create an environment which lends itself to unit tests.

System testing is, however, a completely different problem in distributed systems—simulation works mainly for unit testing. The complexity of interactions, and the number and variety of components at work, generally make simulators uneconomical in such environments. The only economical means is to test software on prototypes, equipped with the appropriate hardware and software probes connected on the microprocessor, the memory and maybe some other critical points. Besides, these tests should be run in interactive mode under a system offering the right kind of debugging facilities, such as traces, breakpoints, snapshots, at the symbolic level.

CURRENT SOFTWARE DEVELOPMENT SYSTEMS

In a system development house, the approach first selected for the development of software for microprocessor systems is to use the existing computer facility. Typically, such a facility is a medium-to-large computer system operated in time-sharing (see Figure Ia). For the benefit of microprocessor software development, existing tools for system and program design, program management and documentation management may be used without problems. However, two questions do have to be answered: What "programming system" should be used? How should testing be done?

In this environment, as far as program translation is concerned, the only possibility is to develop a compiler or adapt an existing one to generate the target code, and to develop a linkage editor and other loader utilities as required by the microprocessor hardware. Further, a microprocessor test simulator has to be developed for unit tests.

When it comes to system testing, programs in load format have to be output on an external medium, which can be used as an input to the prototype system; but, for efficiency, debugging facilities have been implemented on the prototype system to fulfill the requirements stated previously. Though such a development system can work, it has many disadvantages:

1. Development of a cross compiler and of the other programming facilities may be a costly proposition for, after its development, it will have to be maintained and enhanced as required, adapted when new microprocessors have to be used, etc. Even if such a line of translators can be found on the market, adaptation to new microprocessors will generally introduce delays.

2. Simulators have to be developed.

3. Transfer of programs on an external medium is generally not satisfactory because:
   - There is not always a compatible medium on the development system and test system.
   - When an error is found, binary patches will be used on the prototype instead of source corrections, because of the time otherwise involved in updating, compiling, linking and transferring a program. Such a practice is potentially harmful and costly.

4. A debugging facility on the prototype system has to be developed. This may be expensive, if it is to perform symbolic debugging in an interactive mode.

5. When going to other microprocessor compilers, linkage editors, simulators and debugging aids will have to be extensively adapted.

The main advantages are the use of:

1. Existing systems.

2. Existing language-independent software tools, such as design aids, program librarians and documentation aids.

3. If a standard implementation language exists in the user organization, it can still be used on microprocessors at a certain cost, even though it might not give enough visibility to some of the processor functions.

Some of the problems just mentioned, especially in 3, can be alleviated by connecting all consoles to both the development and the test systems. This might be done as shown in Figure Ib, using a concentrator to which both the development system and one or more test systems would be connected via transmission lines. The transfer of programs can then be done automatically, and the user may have access from his console to all resources, whether development or test.

In a development facility equipped with a programmer's workbench, the situation is not significantly better. A cross compiler, a linkage editor and a simulator would have to be developed for a "Programming Machine" (see Figure Ic), leaving the program and documentation-handling facilities on the programmer's workbench, which is essentially a file machine, and does not look like the right system in which to place compilers and simulators unless its configuration is expanded. But then it becomes a conventional system, like those described earlier. The programmer’s workbench does not help to solve the system test problems any better either, since the required debugging facilities again have to be developed on the prototype system.

Another variant of this type of software development system is the Interactive Session Monitor, in which microprocessors are connected directly to the existing development system (Figure Id). Programming tools are run under time-sharing on the development system, and tests can be done either in simulation mode, or by executing the program on the microprocessor. This system again does not solve the problems discussed, since cross-compilers and simulators...
Figure 1—Conventional development systems.
have to be developed, and execution on the microprocessor offers only limited possibilities.

MICROPROCESSOR DEVELOPMENT SYSTEMS

Software development systems offered by microprocessor or independent vendors can be used instead of an existing facility. For example, the software development facilities for the 8080 line of microprocessors found on the INTELLEC system are:

- A "Programming system", including
  - A text editor, to enter, update and inspect text.
  - Compilers, for high-level languages.
  - An assembler.
  - Link/loader facilities.
- A simple file system, to store programs in various formats.
- A debugging system, for unit tests on the system.
- An in-circuit emulator, for testing of a microprocessor program within the prototype environment.

The peripheral configuration consists of a single keyboard console, dual floppy disk drives, and optional printers and other peripherals.

The vendors’ systems support their microprocessors only, but some independent suppliers offer systems which support several microprocessors. Indeed, microprocessor software development systems offer a very adequate range of functions for program development. However, the tools or utilities found on large or minicomputers are entirely missing; there are generally no program library managers (the file systems offered are insufficient in this respect), no documentation aids, no design aids, no project control utilities. Performances are limited—compilations are slow, printing is slow and secondary storage on floppy disks is limited in throughput and capacity.

The advantages, however, are:

1. The programmer has easy access to the system (a ratio of one system to two programmers is generally used).
2. These systems are relatively cheap (in the vicinity of $20,000).
3. In-circuit emulation, which allows one or more programmers to debug on the prototype system, is generally very good.

In large projects involving more than a few programmers, where the volume of code to be produced exceeds 50,000 to 100,000 lines, the investment in development systems is no longer negligible. Diskettes on which programs are stored proliferate and large compilations for system integration hit the limits of the development system, in terms of speed and secondary storage capacity. 4

INTEGRATED CONTROL, DISTRIBUTED POWER

The relatively low cost of microprocessor development systems, the programming facilities they offer, the easy access to the system resources they provide and their in-circuit emulation functions make them hard not to use. Their shortcomings can be overcome by using a two-level system (Figure 2):

1. A centralized level, or "Central System," performing all functions of concern to the project as a whole, and offering services to the programmer where the microprocessor development system is insufficient.
2. A decentralized level, consisting of microprocessor development systems connected to the central system by transmission lines.

Microprocessor development systems

The following functions are performed at this level:

1. Program preparation—Program entry, update and inspection, and program translations, compilation/assembly, link editing.
2. Unit testing on the development system.
3. System testing on the prototype, using the in-circuit emulation facility.

Large volumes of input, such as the initial entry of large programs in source format, are performed on the central system, as is program library management. Typically, the user has his private work library on as limited a number of diskettes as possible, and access in read-only mode to the central library. To prepare his work he may request, from his console, transfer over the transmission line of any number of programs/files, as needed for his own work.

Individual programmer’s programs are included in the central reference library, according to rules depending on the project, the system integration philosophy, and other considerations, but at times that guarantee enough visibility of the state of the work of each individual, throughout the project. A decision to include a new program in the reference library is, however, always an explicit project management decision.

Central system

Each microprocessor development system is seen by the central system as a time-sharing terminal, thus giving the user access to all central system resources.

The central system can be regarded as a file machine and spooling machine.

As a file machine, it manages program and documentation libraries; as a spooling machine, it performs all volume input/output—card input, listing and documentation editing. In fact, no card reader or printer of any sort need be connected on any microprocessor development system—all input/output can go through the central system. In addition, simple keyboard terminals can be connected for functions such as program or documentation entry, update, inspection, which can be done at the central level or elsewhere. Any other
utilities needed, e.g. a source code formatter, a macrogenerator, a test data generator, can be implemented singly on the central system, rather than on the several development systems.

The central system plays very much the same part as the programmer's workbench, except that the latter is a "front-end" processor, whereas here it is the microprocessor development systems that are front-end processors.

**Advantages and disadvantages**

What we have described is not ideal, but it does combine the advantages of centralized systems with those of microprocessor software development systems.

Its residual shortcomings are:

1. The file transfer rate over transmission lines is limited. Most time-sharing terminal throughput is limited to 9600 bauds, which means that the practical maximum transfer rate is around 1000 characters/Sec. This limitation is quite acceptable for a small program, but if a program of 10,000 lines of source code has to be transmitted over a 4800-baud line, transfer will take between 20 and 30 minutes. In practice, this is not really a problem, because:
   a. Modules are stored in source and object format.
   b. Modules are kept fairly small.
   c. Source is transferred only when it is to be modified.

2. The programming languages and debugging systems are those offered by the vendor with the microprocessor development systems. This precludes the standardization of a single implementation language on all microprocessors within an organization—at least at the present time—and makes the client organization entirely dependent on any decision the vendor might make to enhance, abandon or maintain the corresponding products. Though this is not a satisfactory situation, it is not unusual either.

In compensation, however, it may be noted that the system described offers the advantage of being able to use tools, methods and techniques normally used in software development on computers other than microprocessors, thus yielding the kind of technical proficiency and managerial control that the state of the art in computer software engineering permits. Compared to other types of development systems, the flexibility and costs of this system are attractive features.

**Flexibility**

It is generally easy to connect a development system to a system running under time-sharing, so that, if various microprocessors are being used, it will be possible to connect the various corresponding development systems offered by the vendors, or some independent vendor. Further, for operations of general interest not requiring compilation, like...
program or documentation entry, such as gathering information concerning library status, simple time-sharing terminals may be used. With a single central system it will therefore be possible to develop, manage and test programs for microprocessors of various origins.

This set-up, in fact, offers other interesting possibilities. For instance, concerning programming, although the system is clearly geared to perform compilations on development systems, it does not preclude using cross compilers. As reported in Reference 1, software for microprocessor A might be developed using a programming language available on the development system M (B), designed to program microprocessor B. Program entry, compilation and unit testing are carried out on M (B); then a cross compiler on the central system is used to generate code for microprocessor A. The development system M (A), or an independent system, will be used to perform system tests using A on a prototype. Though this process seems fairly involved, it can be made fairly easy in practice, since M (B) and M (A) can both be connected to the central system.

**Costs**

The load generated at the central system in such an application is fairly low, in terms of CPU usage, since most operations are file or terminal operations. A medium-to-large minicomputer should therefore be perfectly suitable. Moreover, actual connect time should be fairly low, since it is essentially determined by the frequency and durations of transfers or other operations requested from the central system.

It is assumed here that a minicomputer in the $300,000 range (hardware only) could provide simultaneous support to 25 to 30 users, and therefore that up to 50 microprocessor software development systems could be supported. The actual cost (purchase price) of the total system, per development station, would therefore be about $6,000, plus the price of the development system itself, which is about $20,000. If it is assumed that one development system can fulfill the needs of two programmers, that the central system has a lifetime of five years, that a development system will be obsolete after three years and that the cost of operating the central system is the same as the cost of hardware, the cost per programmer, per year should be in the vicinity of $5,000.

The service thus provided should be of very good quality, combining those of a minicomputer-based time-sharing system with those of microcomputer development systems. The cost of computer time as assessed above should be compared to figures used for many system developments. Typically, on a system such as a 370/158 where an average of two hours of CPU time per programmer, per month is consumed, the cost per year and per programmer is around $10,000. An overall cost reduction of around 50 percent can therefore be expected with respect to conventional systems.

With respect to stand-alone microcomputer development systems, the concept presented increases the hardware cost by around $6,000 per development system, plus operating costs. But it is pointed out that:

1. Printers on the development systems are no longer necessary, since printing is done at the central site (typical cost of a printer is $2,000-4,000).
2. Extra diskettes are no longer necessary, since a large disk capacity is accessible on the central system (an add-on, double density dual diskette costs around $5,000).
3. Extra services such as library maintenance, documentation aids, etc., and fast I/O devices are accessible on the central system.

**CONCLUSION**

This approach is being progressively implemented at CIT-Alcatel for the development of distributed microprocessor-based switching systems. Initially, INTELLEC systems were connected to CII-IRIS 80 as Remote Batch stations giving access to all development tools available for other developments, then, to improve access conditions and response times, they were connected to a CS 40 (a computer developed by CIT-Alcatel for switching systems), as time-sharing consoles. Replacement of the CS 40 by a minicomputer, to further reduce the costs of the system, is now being considered.

The system described offers distinct advantages with respect to developments on large systems, or on stand-alone microcomputer development systems, although in its present form the overall services offered are limited by the performance of the individual development stations, and by the transfer rates.

Nonetheless, the services offered and the flexibility gained make the system the real solution to the software development problem for the builders of large systems who want to use commercially available development products.

**REFERENCES**