IEEE Computer Society Technical Committee Real-Time Systems Newsletter. February, 1985.
[Abstract. Also presented at the IEEE Second Workshop Real-Time Operating Systems Nov. 1984]

Some Problems in Distributing Real Time Ada Programs Across Machines

R.A. Volz
T.N. Mudge
J.H. Mayer

Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan 48109

Summary

A translator for distributed Ada programs is under study. Such a translator is of particular interested as a tool for the development of real time, embedded systems. The input to the translator would be a single Ada program capturing the logic of the entire system together with special directives mapping portions of the program to the various nodes of the network. The output would consist of a set of inter-communicating Ada programs, one for each node of the system.

There are several advantages of a distributed language. First, by leaving inter-node communication implicit, the programmer is relieved of the burden of explicitly specifying the cross-machine communication required by his application. Our past experience with programming a manufacturing cell in Ada [1,2] has shown this cross-machine communication to be one of the most troublesome aspects of building distributed systems. Second, since the application program is originally written as a single program, it is possible to use a conventional Ada compiler to check for compile-time detectable errors.

The process of distributing any language, such as Ada, involves several steps. First, the definition of the language from the distributed point of view must be verified. Next trial implementation strategies need to be developed. This step should be subdivided into several parts. It is usually helpful to first describe the communications, e.g., a high level protocol, which must take place for each distributed construct. Then actual implementation mechanizations can be developed. Since the goal is to have a distributed *real time* system, the implementation strategies should be analyzed with respect to their real time behavior. Finally, in conjunction with the above, one should look at the run-time support provided to accompany the translator output. The implementation of the run-time support might, in general, involve changes both to the real time operation

system and to the underlying hardware. None of these issues is trivial. We will discuss some of the major issues which arise in the case of distributing Ada.

First, in looking at the definition of Ada in the distributed framework, we have found a number of constructs whose definitions, while being clear in the uniprocessor case, are subject to a number of interpretations in the distributed environment. Two obvious problems are the question of network timing and the apparent uniprocessor bias of shared memory for global variables. There are less obvious but equally interesting examples as well. Two of these are the conditional and timed entry calls. In Ada one task may communicate with a second by calling one of its entries. The caller will then wait until the callee accepts that entry at which time a rendezvous will begin. Since the tasks of real time applications cannot afford to wait indefinitely for a rendezvous, the language provides a means of setting a limit on this type of delay. A conditional entry call is canceled if the called task is not "immediately" available for a rendezvous, while a timed entry call is canceled if a rendezvous does not begin within a specified interval.

In a distributed environment, the calling and called tasks may well be executed on two distinct processors connected in a wide variety of ways ranging from a tightly coupled shared memory connection to a remote connection with relatively slow data rates. How should the resulting communication delays be included in determining whether an entry call must be canceled? One group [3], has decided that they should and that conditional entry calls must automatically fail due to the inherent communication delay. This and other interpretations will be discussed.

The problems are perhaps worse with timed entry calls due to lack of definition of how to take transmission delays into account. Similar problems with transmission delays arise with respect to exceptions. These are also problems arising from cross machine I/O and representation clauses. These issues will be discussed.

Apart from questions about the meaning of Ada constructs in a distributed environment, we are also investigating their suitability to real time programming. One example of this is the task termination construct. In Ada, a subprogram or block cannot be left until all its dependent tasks have terminated. Thus, being able to readily determine the status of a task is crucial since undue delays may prevent a program from continuing after completing a subprogram or block. Unfortunately, the status of a task, i.e., whether it is terminated or not, can depend not only on the status of its own dependents, but also on that of other tasks dependent on its master (its sibling tasks). Specifically a task which encounters a terminate statement may choose to terminate only if its sibling tasks and its own dependents are either completed or likewise able to choose to terminate. Thus, use of the terminate statement means that the status of the task can be determined only on the basis of a global view that encompasses a potentially large number of sibling and dependent tasks. If these are located on a number of different

machines, then assembling this global view on one processor will require extensive and complex use of the communication network which may introduce intolerable delays into the program. This and other questions of the suitability of Ada constructs for real time programming will be discussed.

To provide a testbed for ideas or distributing Ada programs and to help expose major problems a translator for distributed Ada programs is being written. The translator will convert a single Ada program into a set of inter-communicating Ada programs. By including special translation directives (an extension of the Ada pragma), the user explicitly maps his application to the target network. The mapping allows him to assign tasks and task types to specific nodes, thus permitting explicit distribution of the program to meet the real time needs of his system. The translator produces a set of output programs in which all cross-machine communication implied by the user-specified mapping has been made explicit. This includes the automatic construction of agent tasks which are designed to provide each task with efficient access to whatever remote data structures it manipulates and also to aid in cross-machine task synchronization. By producing conventional Ada programs as output, the translator takes advantage of the availability of uniprocessor Ada compilers.

Finally, one of the goals of the above work is to understand the necessary underlying support so that run-time support can be designed (both hardware and software). Some of the obvious issues which must be addressed are the following: the unit execution support must be extended to include at least communications support and network timing. Storage management support might have to address network addressing issues. Tasking support must provide for remote task creation, activation, termination and synchronization. As will be shown, some of these issues, such as task termination, can be quite complex in the case of Ada. Support for exception handling may be particularly difficult since it may not be possible to give an unambiguous time ordering to exceptions that are raised on different machines. The basic operations on typed operations needs to be extended to deal with remote versions of typed objects, i.e., access pointers to off machine entities and representational difficulties. Finally, the predefined packages for I/O and timing will have to be extended to provide network support. Particularly in the case of timing, some innovative solutions may be needed.

References

[1] Volz, R.A., T.N. Mudge and D.A. Gal, "Using Ada as a Programming Language for Robot-Based Manufacturing Cells," *IEEE Transactions on Systems, Man and Cybernetics,* (to appear Sept./Oct. 1984).

[2] Volz, R.A. and T.N. Mudge "Robots are (Nothing More Than) Abstract Data Types," Proceedings of the SME Conference on Robotics Research: The Next Five Years and Beyond, Lehigh University, Bethlehem, Pennsylvania, August 14-16, 1984.

[3] Darpa, A., S. Gatti, S. Crespi-Reghizzi et al, "Using Ada and APSE to Support Distributed Multimicroprocess Targets," Commission of the European Communities, July 1982 - March 1983.