Choosing Ada Tasking Models for Real-time Systems

A recoding project reveals a few of the major problems with writing real-time systems in the Ada language.

By Frank L. Friedman and Paul A.T. Wolfgang

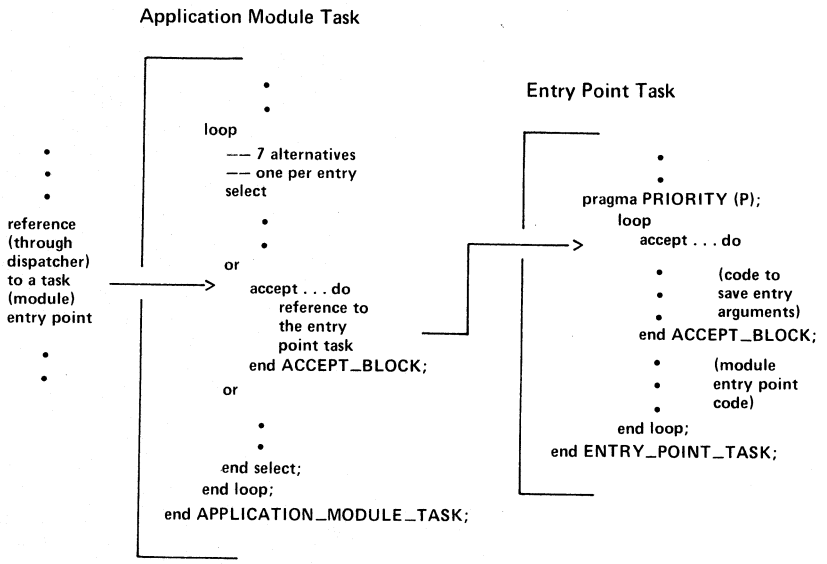
Since November 1984 work has been done at Computer Sciences Corp. (CSC) to recode portions of a large, real-time computer system in the Ada language. The system being studied has been under development at CSC since the early 1970s for use in an embedded shipboard weapons system. It is written in CMS-2 and the Ultra/32 assembly languages and supports multiprocessing, memory sharing, and real-time interrupt services on a special-purpose, onboard computer.

One project involved the construction of the Ada language in skeletal versions of several radar system applications modules and 20 or so executive service handlers. These components were combined into a small mock-up designed to simulate the scheduling and dispatching functions of the executive related to the applications modules and their executive service requests (ESRs).

The original goal of the project was to stay within the framework of the original design for the scheduling and dispatching processes of the executive while at the same time taking advantage of the tasking, priority and preemption support provided by the Ada language. The tasking structure of the resulting system was subsequently reviewed with respect to three factors:

- the accuracy with which it modeled the executive,
- the degree to which the Ada tasking, priority, and preemption mechanisms were utilized and
- the appropriateness of the choice of this tasking structure for use in modeling the system being studied.

As a result of this review, the tasking structure of the system was rewritten.



Tasking schematic for entry point reference in application moduel tasks

***Tasking schematic** for entry point references in application module tasks.*

Basic Work Units

Both the weapons software system and the Ada language address modularization and reusability, controlled data access, and specification and use of tasks (called modules in the weapons system) as the basic system work units. The weapons software is highly modularized and consists of many small, functionally complete components organized in a hierarchical system structure. Module interfaces are carefully controlled through the use of well defined and documented message packets and a combination of code and style conventions that limits access to several levels of common data. The modularized, hierarchical system structure, combined with the consistent use of coding, style, and naming conventions, have produced a system that is easy to read, considering the low level of the implementation languages. The system structure and the implementation conventions have also contributed to a high degree of component reuse as the weapons software system has evolved.

The reusability concept was a major motivating factor in the design of the Ada language. The data abstraction facilities support the encapsulation of data and operations in a manner conducive to ensuring clearly defined inter-

Frank Friedman is associate professor and department chairman at Temple University's department of computer and information science and a senior computer scientist at Computer Sciences Corp. defense systems division. Paul A.T. Wolfgang, formerly lead scientist at Computer Sciences Corp., is the current manager of software engineering at Boeing-Vertol Co.

faces while hiding implementation details. The Ada language was also designed to handle concurrency, providing an explicit tasking mechanism for representing interprocess communication and control algorithms without forcing the programmer to step outside the high-level language environment.

Thus, the Ada language has the potential of supporting the behavior of the weapons systems and its executive. The primary goal of this work is to ascertain the extent to which features of the current executive execution model can be mapped into the Ada language task model. Also being studied is the extent to which the Ada language run-time environment must be augmented to support specialized features of the executive such as changing priorities, task and entry point priority assignment and preemption restrictions.

The scope of the experiment is limited to an analysis of a small subset of weapons system tactical applications modules and of the scheduling and dispatching of these modules. System functions related to initialization, periodic rescheduling, priority assignment, preemption, common data access and module communication were studied and recoded in skeletal form in the Ada language.

Executive Overview

The executive is divided into two major functional units: an executive service program that provides the nucleus of executive services to control the CPU, input/output channels and memory; and a dependent executive program that provides user-dependent services for a particular tactical application. These services include system initialization, interrupt handling, scheduling and dispatching, memory management, message processing, input/output and device processing, error processing and recovery, inter-computer communications, common service routines (such as mathematical functions), and performance measurement and debugging support.

Five tactical applications systems are serviced by the executive. The basic work unit of these systems is the single-function tactical applications module. These modules perform functions related to carrying out the system mission, including the acquisition, processing, evaluation and display of tactical data.

Tactical application modules may be scheduled for processing at any one of seven entry points, one for each of the primary processing activities performed by the module. The entry points for all modules are the same: one each for initialization, message processing, error processing, successor processing, buffer complete functions, channel complete functions, and periodic processing. For most modules, however, one or more of the seven entries will be undefined (null) and therefore unable to be scheduled.

The scheduling and eventual execution (dispatching) of all module entry points is managed through a priority schedule queue (PSQ). The executive

uses a two-tier priority scheme with a preemption priority (the major priority) governing execution preemption and a scheduling priority (the minor priority) used for maintaining the order in the PSQ. A schedulable module entry point may be dispatched (subject to preemption restrictions) when it reaches the front of the PSQ. Priorities associated with a module entry point may be altered at execution time via requests from that module or others.

The scheduling of a module's initialization, error processing, message processing and successor entry points may be done by other tactical application modules via ESRs. The buffer and channel-complete entries are scheduled directly by the executive, not through requests from other applications modules. Module initialization is scheduled directly by the executive whenever system initialization is required.

Periodic entries are scheduled directly by the executive each time a countdown clock interrupt occurs. Schedulable periodic entries are stored in a separate periodic queue (PQ) ordered by "next-time-to-go." At each countdown clock interrupt, all ready periodics are deleted from the periodic queue and entered in the PSQ, subject to conditions concerning duplicate periodic entries in the PSQ. If a periodic entry is reschedulable, a new time-to-go is determined and the entry is reinserted into the periodic queue. The reschedulable and multischeduling status of a module's periodic entrance may be changed via requests to the executive.

```
loop
   accept RUN (...) do
   .
   . save arguments (if any)
   .
   end; — accept-do
   — ensure preemption is allowed
   — ensure no active entry with same
      module id
   — dispatch entry point if OK
   SCREEN_FOR_RUN
   (..., DISPATCH-OK);
   if DISPATCH_OK then
   DISPATCH_ENTRY (...); end if;
end loop;
```

The reschedule and multischedule attributes of a periodic entry are used to determine the conditions for reinsertion of a periodic entry into the periodic queue. Each time a periodic entry is scheduled for execution (deleted from the periodic queue and entered into the PSQ) these attributes are evaluated. If the reschedule indicator is off, the entry is not reinserted into the periodic queue. If the multischedule indicator is off, the entry is reinserted only if the reschedule indicator is on and the entry is not already present in the periodic queue.

Preemption of an executing module entry point may occur only when an entry point of another module with a higher preemption priority has reached the head of the PSQ and the system is in a preemptive state. Preemption is not allowed if the preempting entry is from the same module as the currently executing entry, or if it is from a module with a previously activated but preempted entry waiting for completion.

Scheduling and Dispatching

The main task components of the original mock-up are an executive task, an external environments task, and a collection of applications modules tasks.

The executive task serves as the parent for all ESR processing modules. It contains an entry point for each supported ESR and interrupt and operates at the next-to-highest priority level in the system.

The external environments task provides a coarse simulation of interrupts external to the portion of the executive being modeled. This task provides a mechanism for processing periodic entry points—removing from the periodic queue and inserting into the priority schedule queue—and for the random insertion into the PSQ of other schedulable entries. The external environments task runs at the highest system priority level, and has a programmed delay permitting other tasks to execute when the external environments task has completed its work.

The collection of applications module tasks consists of eight tasks for each module: a top level task with a family of seven entry points and seven lower level tasks—one for each entry—which model the functions to be performed at each entry.

In developing this model, the first problem to be resolved was the assignment of a priority value to each entry point of a module, a feature that was an integral part of the scheduling and dispatching process of the executive. The Ada language does not allow priorities to be assigned directly to task entries, so a tasking model was required that permitted priorities to be assigned to each system component responsible for entry point processing.

An Ada language model was chosen consisting of a single top-level task with a family of seven entries. Underneath this task, another level of tasking was added that allowed each scheduled entry point to be assigned a unique, although static, priority. In this model, each schedulable entry in a task consisted solely of an accept-do block containing a statement referencing another task that actually carried out the functions of that entry. This task was assigned a priority, P, that reflected its relative level of importance in the executive.

Limitations of the First Model

Unfortunately, the executive scheduling and dispatching scheme could not be completely and accurately represented within the first Ada language model. Aside from the dynamic priority requirement, which could not be handled within the model, other difficulties were encountered with respect to the executive priority and preemption schemes.

One problem was caused by the two-tier priority scheme (with 66 different levels of urgency) used by the executive. The Ada language supports only a single-tier priority scheme. A function was required in order to map the two-tier priority scheme to the single level of Ada and vice-versa.

The executive preemption scheme presented another source difficulty in the design of the Ada language mock-up. The executive allows one module entry to preempt another only if the executing module is in a preemptive mode, and then only if: the preemption (major) priority of the waiting module entry is higher than the preemption priority of the executing module and the waiting module entry is not in the currently executing module nor is it in any module that has been preempted.

Only one module entry can be active (executing or temporarily suspended) for each preemption level (1 through 4) and for each module. Those conditions require that the Ada language model of the executive keep track of the preemption level and module identification for all active module entries. That is a step in the wrong direction, since one of the primary goals in the project is to use the Ada language task model as much as possible to keep track of the environment and status of preempted tasks so the entire scheme would be transparent to the programmer.

First Model Design Review

The selection of a tasking structure for an embedded computer program is one of the more important system design decisions that must be made. The Ada language tasking and priority features offer a variety of tasking strategies that might be used to any given problem. While data structuring and packaging considerations should receive emphasis during the design of a system, tasking strategy issues also should be dealt with early in the design process.

The current literature on tasking in the Ada language provides little guidance with regard to tasking strategy choices. Most tasking examples appear to be designed to maximize concurrency as constrained solely by an abstract problem description that contains little information about performance requirements and underlying hardware architecture.

The use of the Ada language and related design and implementation methodologies is intended to reduce the impact of hardware related constraints upon all phases of the software development process, thereby providing greater adaptability and portability of the Ada language software. Given the current and near-future state of the Ada language compiler and run-time-support technology, it seems unreasonable to continue to study the design of real-time system models without consideration of the hardware and performance constraints that must be faced. Despite recent advances and continuing research, these constraints in on-board systems are evident, and a high degree of parallel computation seems remote.

The structure of the executive being modeled was dictated by severe memory constraints imposed by the host computer. As illustrated by the preemption model description at the end of the previous section, the current system does no time-slicing, and pre-

emption is highly restricted, thereby minimizing the frequency of process state changes and the amount of memory required by these changes.

The review of the first scheduling and dispatching model was conducted with those issues in mind. The Ada language's tasking support was useful in maintaining the desired tasking and management relationship between the executive task, the external environment task and the collection of applications modules tasks taken as a whole. However, it could not be used to manage the scheduling and dispatching of the individual module entry point tasks. It was necessary to write explicit Ada language code to manage module entry points.

This Ada language tasking model did not provide an accurate representation of the parallelism of the executive. The model allowed for the concurrent activation of all module entry points, while the executive allows for only a single active entry per module per preemption level. The Ada language model also precluded the dynamic assignment of priorities to each module entry, since the Ada language does not allow for either dynamic priorities or entry priorities. The model carried with it the overhead of excessive tasking, with minimal benefit in terms of the utilization of the tasking support provided by the Ada language. While the model may have been of interest because of the theoretical level of concurrency that it provided, little of this concurrency could be realized in practice because of the constrained hardware and system design environment.

Another Tasking Strategy

As a result of this assessment, it was decided to rewrite the system with each applications module represented as a single package encapsulating its own data structures and seven top-level procedures each representing a single module entry point. The only tasks that remained were the executive task, the external environment task and four new preemption level tasks. Each of those tasks was responsible for controlling the scheduling and dispatching of all module entry procedures having a given preemption level.

This second model mirrors the existing system much more closely than the first and has the added advantages of less tasking overhead and a simpler component structure. It also allows for dynamic priority assignment, since the changing and evaluation of entry priorities is now table-driven, as is the version of the executive being modeled.

The new Ada language model permits the utilization of the Ada tasking support to a greater extent than before. Each of the four preemption level tasks (named PREEMPT-LEVEL-1 through PREEMPT-LEVEL-4) operates at a different priority level (ranging from PRIORITY'LAST-2 for the task servicing preemption level 1 down to PRIORITY'LAST-5 for the task servicing preemption level 4). Since the dispatching and execution of a module entry point of preemption level P (1

$\leq P \leq 4$) takes place under the control of task PREEMPT-LEVEL-P, a higher-level entry point cannot be preempted by a lower-level one. Furthermore, because each preemption-level task is implemented using a conventional loop/accept construct, a rendezvous with the task that services preemption level P can only occur when no level P entry point is active.

The Ada language tasking model guarantees that there can be no more than one active entry for each preemption level and that a higher-level entry cannot be preempted by a lower-level one.

Other Modeling Issues

In addition to the tasking issue, many other problems arose from our efforts to model the weapons system in the Ada language. Most of those problems relate to the structuring of data, and the degree of hardware-dependent detail required to achieve adequate data definitions with respect to numeric accuracy, time and space efficiency, intersystem and intercomputer communication and meaningful representations of the abstract structures being modeled.

Eventually enough should be learned about the Ada language, its compilers and run-time system implementations for specific hardware, so that convenient and efficient solutions may be found. Appropriate solutions will become apparent only when the entire system is redesigned with the Ada language, portability, adaptability and maintainability in mind. Perhaps appropriate solutions will be possible only in assembly language. Whatever the case, the path to the successful and effective use of the Ada language may be more difficult for the real-time, embedded systems domain than for any other applications area. That in itself seems contradictory, considering the original purpose for the development of the Ada language.

Problems Remain

Considerable work remains to be done with the current system mock-up. Some of this work is essential for the completion of the evaluation of the use of the Ada language in implementing the scheduling and dispatching functions of the executive. Other work involves the executive functions not yet addressed by the current limited mock-up, such as alternative strategies for implementing the various message communication techniques, error processing and exception handling and input/output and channel/buffer processing.

The Ada language potentially can support much, but not all, of the underlying behavior of the executive. Since we are examining only a small part of the executive, it is certain that we have uncovered a few of the major problems associated with writing real-time systems in the Ada language. The planned evaluation of additional weapons system components for input/output processing, interrupt, error, and message handling, common service routines and other implementation-dependent issues will most likely reveal

additional problems.

It might be argued that most, if not all of these problems, are unique to the executive, specifically to its original design. It is possible that all of them can be eliminated by redesigning the system for eventual recoding in the Ada language. At this point, we are not convinced. We believe that the Ada language tasking model, including the preemption/priority scheme and the termination mechanism, may not be sufficient to support a typical real-time tactical embedded computer system. We are concerned too, about the efficiency with which the Ada language run-time systems will support recoded executive functions and about the portability of such systems. Others (see references to NAVSEA 83 and Payton 83 below) have similar reservations. Unfortunately, an Ada environment appropriate to realistic studies in these areas is not likely to be available in the near future. DE

BIBLIOGRAPHY

[Booch 83]
Booch, Grady, Software Engineering with Ada, Benjamin/Cummings Publishing Company Inc., Menlo Park, CA 1983.

[Booch 84]
Booch, Grady, "Dear Ada," ACM Ada Letters (III,5), March, April, 1984, pp. 29-32.

[CSC 85]
Computer Sciences Corporation, "Observations on Modeling a Real-Time Embedded Computer System Using Ada," IR&D Technical Report No. SP-IRD 4, Moorestown, N.J., June, 1985.

[Intermetrics 83]
Intermetrics, "An Analysis of the Problems Associated with the CMS-2 to Ada Transition," Report 0967-LP-598-9720, prepared by Intermetrics Incorporated, Bethesda, MD, for the Naval Sea Systems Command, Washington, DC, June, 1983.

[NAVSEA 83]
NAVSEA, "A Plan for the AEGIS Transition to Ada," Report 0967-LP-598-9730, prepared for the Naval Sea Systems Command, Washington, DC, June, 1983.

[Payton 83]
Payton, Teri F., and Michael J. Horton, "Study Report on the Transition of RNTDS to Ada," Report FR(A)-3020, prepared by Systems Development Corporation, Paoli, PA, for the Naval Sea Systems Command (Code 06L3), Washington, DC, 31 May, 1983.

[Softech 83]
Softech, "CMS-2 to Ada Transition Plan," Report 0967-LP-598-9750, prepared by Softech Incorporated, Falls Church, VA., for the Naval Sea Systems Command, Washington, DC, August, 1983.

[Softech 84]
Softech, Slides for Course L401, "Real-Time Systems in Ada," Parts 1 and 2., 1984.