IMPLEMENTATION OF AN ADA* REAL-TIME EXECUTIVE - A CASE STUDY

James D. Laird
Dr. Bruce A. Burton
Mary R. Koppes

Intermetrics, Inc.
Aerospace Systems Group
5312 Bolsa Avenue
Huntington Beach, California 92649

ABSTRACT

Current Ada language implementations and runtime environments are immature, unproven and are a key risk area for real-time embedded computer systems (ECS). This study provides a test-case environment in which the concerns of the real-time, ECS community are addressed. A priority driven executive is selected to be implemented in the Ada programming language. The model selected is representative of real-time executives tailored for embedded systems used in missile, spacecraft, and avionics applications. An Ada-based design methodology is utilized, and two designs are considered. The first of these designs requires the use of vendor supplied runtime and tasking support. An alternative high-level design is also considered for an implementation requiring no vendor supplied runtime or tasking support. The former approach is carried through to implementation.

* Ada is a Registered Trademark of the U.S. Government (AJPO)

INTRODUCTION

Since the inception of the common DoD High Order Language (HOL) effort in the mid-70's, the Ada programming language has remained a cornerstone of the government effort at producing software in a cost-effective manner. Validated Ada compilers are becoming available on a variety of different computers with at least 17 validated compilers now available and more slated for validation during the current year. There are currently 37 different defense programs using Ada, and this number is anticipated to exceed 120 during the next four years[1]. While this progress is encouraging, the success of the Ada language in meeting the needs of specific applications will hinge on the consideration of the potential risks that face the implementors of a given system.

This process of risk identification should be followed by development of risk minimization and avoidance strategies tailored to meet the needs of the system. The emphasis of this paper is in

the area of technical risk identification and resolution for real-time ECS applications. While the Ada programming language is intended for real-time applications, current compilers and runtime systems are unproven for these types of programming efforts. Consequently, the impact and implications of using the Ada language and Ada-oriented methodologies in embedded real-time development efforts should be assessed. While it is necessary to examine how well and to what extent the built-in real-time features of the language meet the needs of ECS applications, additionally, we must re-evaluate the standard approaches to solving real-time problems in light of the new capabilities and assess the impact, if any, on the way we design and implement these solutions in software.

SCOPE

Perhaps the major consideration with regard to the use of the Ada programming language for real-time ECS applications is the cost of doing so in terms of memory and processing overhead. The relative costs associated with the use of Ada and its real-time features is especially relevant to small embedded computer system applications given the physical and temporal constraints imposed on these types of applications. The determining factor in the decision to utilize a particular high order language (HOL) feature is often the efficiency of its implementation. It is important to know what the utilization of Ada with its real-time tasking primitives, representation specifications, exception handling, and various other features translates to in terms of program size, speed, and efficiency. The ability to selectively include runtime support and its resultant overhead for these features on an "as needed" basis is another important consideration. During the course of this investigation, answers to fundamental questions such as these were sought.

BACKGROUND

It is important to stress the significant conceptual differences between the two approaches investigated with regard to this case study implementation of a priority driven Ada executive. Figure 1 serves to illustrate the alternative approaches and concepts and their implications for the developer of an Ada executive.

The terms O.S., executive, and runtime support or system (RTS) are often used rather loosely when ECS topics are discussed. The ambiguity of this terminology in the ECS environment is primarily due to the overlap in functionality provided by different implementations for different applications. An application residing on a bare machine may interface with software providing minimal scheduling and memory management. This software is often referred to as an "executive" or runtime

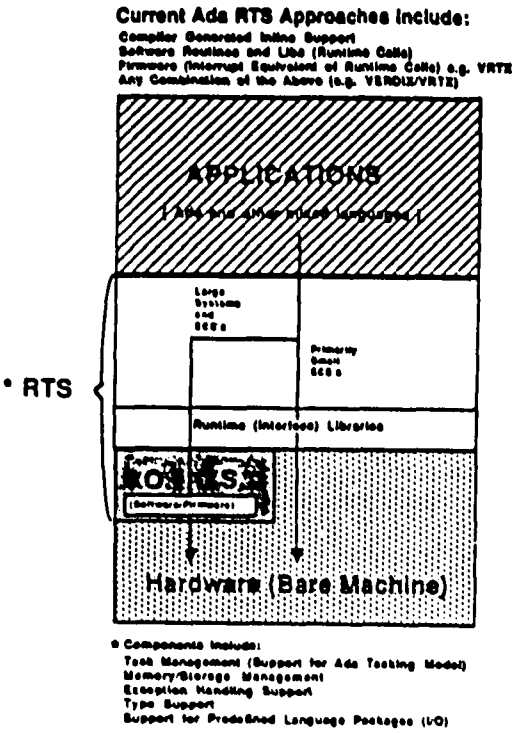

FIGURE 1
RUNTIME SUPPORT (RTS)
APPROACHES

kernel whereas the same services provided on another system may be obtained from software referred to as an O.S. The primary difference in terminology is attributable to the variety and nature of the services provided by the support software in question. The more minimal the services provided, the more likely that the terms runtime support, runtime kernel, or executive will be applied. True operating systems in the strict sense are distinguished by two major factors. They are typically developed independently of any compiler-/applications software and are acquired independently rather than as a part of a given compiler system or package.

The other major distinction is in the comprehensiveness of the services provided by an O.S. for the target machine; services that may be targeted and utilized by a variety of differing applications and tools as well as many different compiler systems. The minimal runtime support for applications developed under a single compiler system may interface to, and utilize, the comprehensive services provided by an O.S. Therefore, the RTS for an ECS can be thought of as providing the minimal required subset of O.S. services needed for a given application. As stated, this minimal subset can be provided by direct access to the underlying machine or through the utilization of the services provided by an underlying comprehensive O.S. The former case is the most typical for embedded computer systems. The term "executive" is most often used to refer to that part of the RTS that performs the basic scheduling and memory management. Other portions of the RTS may include I/O control, timer/clock management, and a certain amount of system level runtime error and interrupt trapping.

The RTS of an ECS supports the execution of application programs and the programming language features utilized to develop those programs. As illustrated in Figure 1, this support can be implemented in hardware, microcode, through direct calls to an O.S., through the use of runtime support libraries, or by compiler

generated (in-line) code. The operating system and RTS needs of small embedded computer systems are typically modest. All that such small ECS targets usually require is an "executive" consisting of little more than a basic scheduler, memory manager and some type of I/O manager or controller. Obviously, different applications may have specific needs relative to memory management, I/O, or clock services which will be reflected in the "executive/-O.S" software.

APPROACH

This paper addresses two basic options or approaches to the implementation of an Ada executive and briefly discusses ongoing as well as proposed work in a third area of related investigation. The first of these approaches is explored in depth (through to implementation) and consists of a combination of a "pseudo executive" or scheduler at the applications layer in concert with vendor supplied executive software at the runtime system level. The obvious benefits of such an approach - imposing an additional layer of control upon the runtime system scheduling mechanism - include ease of portability, and relative target independence with respect to the underlying scheduling algorithm at the RTS layer. These benefits as well as the tradeoffs in overhead and consistency from implementation to implementation will be discussed in detail.

The second option is explored at a high level only. This alternative, termed the bare machine approach, is consistent with the traditional approach to avionics-based executives and is considerably more limited in scope than the first in the sense that it assumes no underlying vendor supplied runtime support. This executive performs all necessary support for the execution of user jobs or "tasks". However, this approach is significantly more restrictive than the first with respect to the nature of what constitutes a "task" as well as to the use of certain Ada language features involving both the Ada tasking model and dynamic memory management and certain other real-time aspects of the language.

The third option is considered only in terms of current and ongoing investigative work and proposed future studies based upon the results of past investigations. This approach diverges from the others in that it proposes a migration to the runtime system layer in order to probe the issues of efficiency and risk reduction for real-time Ada applications. This option emphasizes the tailoring and optimization of the executive functions provided at the RTS layer.

A multi-phased approach beginning with a requirements specification was utilized for the design and development of the priority driven executive. The functional capabil-

ities that were to be provided were extracted from an existing avionics executive implemented in a combination of FORTRAN and Assembly language. It was determined that these same functional capabilities would be provided within the executive being implemented in the Ada language.

While providing substantially the same functionality, the Ada equivalent constituted a complete re-design utilizing Ada concepts and features where possible. For this reason, the Ada executive posed some unique problems from the outset with respect to use of the new Ada concepts and features such as the Ada tasking model. These issues are addressed in the RESULTS section of this paper.

The Ada priority driven executive was to provide facilities for the creation of active tasks via a scheduling mechanism. The scheduling mechanism would provide time-dependent scheduling capabilities, precision timing of task activation as measured by time base generated (TBG) epochs, and signal dependent scheduling capabilities. The Ada priority driven executive would perform prioritized tasking and would have the option of enabling and disabling interrupts. The capability to directly connect to a real-time clock interrupt would be provided. In the absence of such a facility, the real-time clock interrupt would be simulated with the smallest granularity possible. In short, the Ada priority driven executive was required to be a real-time, multi-tasking process manager with interrupt handling and both cyclic and asynchronous scheduling capability.

Integral to the design of the Ada priority driven executive was the selection and application of a state-of--the-art, Ada-based design methodology. A somewhat novel design approach was selected that was based upon Object Oriented Design[2] with enhancements and modifications specific for real-time embedded systems[4]. The methodology derived was termed Real-Time Object Oriented Design (RTOOD) and drew upon another real-time, systems--based design methodology called Design Approach for Real-Time Systems (DARTS)[5]. The steps utilized in this hybrid methodology are outlined in Figure 2.

I. Definition/statement of the problem

II. Informal strategy (Modified specification)

III. Identify objects and attributes

IV. Identify Operations

V. Identify concurrency *(DARTS)

Decomposition into tasks/packages based on:

The asynchronous nature of major transforms
- sequential vs. concurrent

specifically:

Vo dependency
time critical functions
computational requirements
function cohesion
temporal cohesion
periodic execution

VI. Establish the interfaces

VII. Implement the operations

* (DARTS)
Design Approach for
Real-Time Systems

FIGURE 2
REAL-TIME OBJECT ORIENTED DESIGN (RTOOD) METHODOLOGY

Similarly, a high level design was developed for the alternate approach - termed here "the bare machine approach" - to the development of an Ada executive. The "bare machine" model implements its own concurrency through the executive while disallowing the use of the Ada tasking model per se as well as any difficult, and potentially risk-prone, dynamic storage management. The potential benefits and risks of each of these approaches was examined with the former approach being carried through to implementation and limited utilization.

RESULTS

I. ADA EXECUTIVE WITH VENDOR RUNTIME SUPPORT

The capabilities of the FORTRAN/Assembly language implementation and the Ada language implementation are summarized in Table 1. The Ada language version consists of two major components - the program code and the vendor supplied runtime system. In both implementations the scheduling primitives are provided by the executive, but the ultimate responsibility for cyclic/acyclic task scheduling lies with the user (application) tasks. Note, however, that the task interleaving and task waiting in the Ada language version is strictly under the control of the Ada runtime system and not under the control of the executive as in the FORTRAN/-Assembly implementation. Furthermore, although tasking could be prioritized dynamically (changed) in the FORTRAN/Assembly implementation, priorities at the runtime system level are static in the Ada language version.

Functional Summary

Figure 3 depicts the major functional components of the Ada equivalent prototype developed for the case study investigation. The major distinction between the Ada implementation and the FORTRAN/Assembly model depicted in Figure 4 involves the interaction of the Ada runtime system with the priority driven executive functions.

TABLE 1 FORTRAN/ASSEMBLY VERSUS Ada IMPLEMENTATION

CAPABILITY	FORTRAN/ASSEMBLY EXECUTIVE	Ada EXECUTIVE	Ada RUNTIME SYSTEM
CYCLIC/ACYCLIC TASK SCHEDULING	Provided	Provided	
TASK DISPATCHING	Provided	Provided	
TASK INTERLEAVING	Provided		Provided
TASK WAIT	Provided		Provided
PRIORITIZED TASKING	Provided	Provided	Provided
TBD INTERRUPT HANDLING	Provided	Provided	

While the FORTRAN/Assembly model managed all state transitions for user tasks from inactive to executing and all information associated with these state transitions, the Ada implementation utilizes the Ada runtime support system (for the tasking model) to manage the active processing phase of any user task as well as the body of information associated with a tasks' active execution. Specifically, the Ada runtime

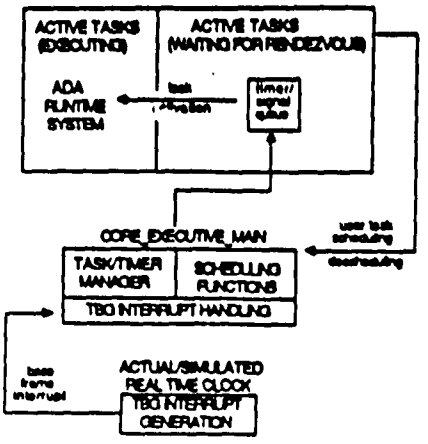

FIGURE 3
ADA PRIORITY DRIVEN
EXECUTIVE FUNCTIONAL
SCHEMATIC

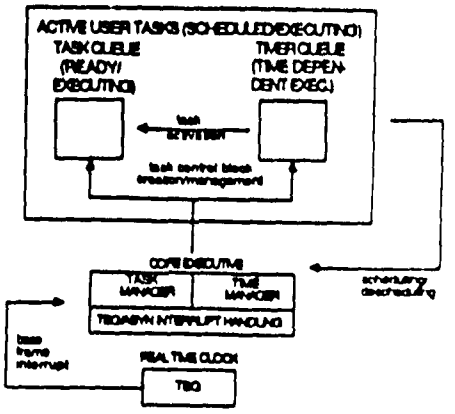

FIGURE 4
FORTRAN/ASSEMBLY
EXECUTIVE FUNCTIONAL
SCHEMATIC

system manages the interleaving or time-slicing of concurrently executing user tasks and is responsible for management of the associated task activation information. The start of a user tasks' scheduled execution phase is strictly under the control of the Ada priority driven executive at the applications layer, yet, the management of the transfer of control between any number of concurrently executing user tasks is by definition under the control of the vendor supplied Ada runtime system.

To satisfy the requirement for a cyclic capability, the executive was required to have some method for specifying fixed-rate scheduling. This was provided on two levels. In keeping with the scheme utilized in the original model, the facility for scheduling a task for execution is provided. Active tasks currently executing may therefore utilize this facility to re-insert themselves into the schedule for future execution, or this may be done by some other active user task.

In the original model a voluntary, non pre-emptive scheduling scheme was utilized among the user tasks that enforced the notion that no transfer of control or context switching among tasks could occur unexpectedly. Bearing in mind that within an Ada environment the underlying operating or runtime system utilizes another level of scheduling for the interleaving of currently active tasks, a task prioritization scheme among these tasks is then required to enforce the notion that a particular task is incapable of having its scheduled execution interrupted once it begins.

In short, we have a scheduling scheme at the user task level to specify fixed--rate triggering of a tasks' processing and the Ada pragma "PRIORITY" enforced at the underlying operating or runtime system level to ensure uninterrupted completion of that processing.

The major potential point of failure with respect to this type of approach to task scheduling at the applications level is at the underlying runtime system level. The issue is one of consistency from implementation to implementation with respect to time slicing of concurrently executing processes of equal priority. While fixed rate triggering of task execution can be guaranteed via a combination of algorithmic control, prioritization, and interrupt handling through the "psuedo executive", no such guarantee can be made with respect to the method of time slicing utilized by the underlying runtime support for concurrent tasks of equal priority. This will vary from implementation to implementation although adhering to the so-called "FAIR" requirement dictated by the language specification. Given the stringent nature of typical ECS performance and reliability requirements, this potential inconsistent behavior across implementations could pose a significant risk.

Static prioritization of Ada tasks may be a problem in some instances of task scheduling or interrupt handling since external events often dictate a need to dynamically change priorities. The Ada rendezvous occurs in a first in, first out manner using a queue structure for multiple entry calls issued for any given task entry point (ACCEPT statement). There is no way to reorder and influence the position a calling task may occupy in such a queue. It is possible that with dynamic task prioritization this could be programmer controlled.

Efficiency: Space and Time The FORTRAN/ Assembly language implementation used as a model in this case study was coded in a little over 1 K (bytes) of memory and accounted for somewhat less than two percent of the entire system. While the entire Ada system consisted of just over 700 lines of code, the space requirements varied with respect to the host machine. The Ada version required anywhere from 27 K to 38 K bytes of memory for the applications code alone. The runtime kernel on one machine imposed an additional penalty of 200 K bytes to utilize the Ada tasking model. It should be noted, however, that the executive was developed for functional realism and was not optimized for minimal program size. The runtime kernels were large, as much as 200K bytes, but the runtime kernels were intended for a main-frame environment, not a typical ECS application.

The significant lessons learned were in what options were available to optimize the size and speed of the execu-

table image. Significant savings - approximately 100K - were available via a selectively loadable tasking kernel in at least one implementation while other options resulting in savings were no runtime checking (1-2K savings), and no debugging instrumentation (5K savings). In one particular implementation, the option for space optimization was offered yet yielded no appreciable difference in the size of the executable image.

While there is no strict linear relationship with respect to overhead between host and ECS environments, the significant savings realized through configurability within the host environments has significant positive implications for ECS environments where efficiency constraints are paramount.

It was found that the total storage penalty to include a minimal exception handling capability within each Ada program unit was on the order of 4-5 percent of the total program storage while the cpu overhead to invoke an exception handler ranged from 30-500 microseconds. This represents an acceptable cost in either a host mainframe or embedded environment.

The overhead in terms of time to utilize the rendezvous mechanism within the host environment was rather high, being approximately 11-12 milliseconds. Given the relatively rapid frame times of many real-time applications (on the order of 40-100 milliseconds), a feature that uses approximately one tenth of the frame time poses serious risk[3]. However, based upon current investigations with Ada for embedded 16 and 32 bit targets, the case can be made that this is a problem somewhat localized to the mainframe environment.

II. THE BARE MACHINE APPROACH

The alternate design approach proposed in this study for the Ada priority driven executive (see Figure 5) is intended for a bare machine environment with no resident operating system nor any vendor supplied Ada runtime support. The design of such an executive raises some important issues with respect to validation when considering what must be provided to support the execution of an Ada application on such a bare target. The implications of the traditional model of an executive, such as the original FORTRAN/Assembly language implementation used as a basis for this study, are considered.

This approach differs greatly from that which utilizes an underlying runtime system. This approach implies that beyond the generation of native machine instructions from the HOL by some generic translator or compiler, it becomes necessary to provide programmer supplied support for any HOL language features not directly implementable through primitives on the bare hardware. It therefore becomes the task of the

runtime supervisor or executive software to provide this underlying support for things such as concurrency or multi-tasking, I/O, dynamic storage and memory management to name a few. In addition, this executive must not, in turn, rely on some underlying support for its own execution.

Validation is certainly an issue with respect to this kind of subset Ada approach. While recognizing the incompatibility of this approach with the notion of validation, we choose not to address the topic in any detail other than to acknowledge the conflict. Our focus is on technical risk identification and minimization.

The design of this bare machine executive was purely hypothetical and no specific embedded target was selected. For that reason, only a high-level design was iterated. Currently, typical vendor supplied Ada runtime support packages facilitate things such as: system elaboration or initialization, task communication and scheduling, exception handling, interrupt, I/O, and type support. The amount of overhead varies with each vendor's implementation. The design proposed is for an Ada executive function that would minimally support the execution of other Ada software constituting jobs or "tasks". However, the Ada tasking model is not supported by the proposed subset Ada implementation for a bare ECS target.

As in the traditional model, concurrency is achieved via the executive utilizing a non pre-emptive, voluntary context switching mechanism. Control over scheduling is therefore explicit and known to the programmer. In addition, any dynamic data or storage management is restricted to that which supports the execution of the executive functions only.

It must be noted that the notion of an "all Ada executive" at this level is fallacious. A certain amount of privileged accessing of register and stack contents by the executive functions to facilitate the basic context switching and memory management would be required. This is not directly achievable from within the Ada language. Therefore, a component of the executive software (e.g. the Control_Transfer_Package) would by necessity be implemented in a lower level programming language. In current commercial Ada runtime systems for embedded targets such as the 1750A, this accounts for approximately two percent of the vendor supplied runtime support. Ada packaging concepts facilitate the encapsulation and isolation of such machine context sensitive components.

The rationale for the approach to concurrency presented is straightforward. While explicit context switching can be considered risky, it has certain potential benefits. It avoids the necessity of excessive locking since the programmer knows

exactly when context switches are to be performed. Another benefit is realized when a high priority event occurs that must be handled rapidly as is the case in many real-time systems. While handling such an event, it may be deleterious to release the processor. Finally, the avoidance of unnecessary context switches and/or checking results in greater efficiency[8].

Admittedly, however, it is reasonable to question the feasibility and advantages of using Ada without its tasking features and other real-time components versus using any other high-level programming language. It should also be noted that, with some re-working of the design, there is nothing to explicitly prevent the use of the Ada tasking model and rendezvous concept, provided that the necessary runtime support is supplied at an acceptable cost in memory overhead and execution efficiency. This is the motivating concept driving our current and future investigations with respect to Ada real-time systems and will be discussed in the following section.

<u>Current and Future Investigations</u> The rationale for an approach such as the bare machine option is that given the present state of tasking support in an environment that supports full Ada tasking, exception handling and other HOL features, the resultant program size may be unsuitably large for an embedded application. While the applications level strategy and the bare machine approach represent two available options, an additional alternative exists that holds some promise for the design of compact, efficient real-time systems and is the focus of our current and future investigative work. This consists of a migration to the RTS layer in pursuit of optimization and risk reduction at this level while maintaining the complete (or nearly complete) functionality of the language. The focus is on tailorable, configurable runtime support for the design of efficient real-time systems in Ada.

It is highly likely that the full functionality of the traditional model of a priority driven executive can be achieved in this manner by minimizing the role of a programmer supplied executive and relying on the efficient implementation of the Ada tasking model at the operating or runtime system level. While it may still be necessary to provide customized runtime/executive support, this can be provided primarily through tailoring of existing systems at the RTS level to meet specific performance requirements rather than exerting additional control at the applications layer.

Our current efforts are focused foremost on proof of concept - that we can design and implement fast, compact, efficient, real-time systems in Ada - with a secondary emphasis on the validation issues. The steps we have identified as being necessary

to the success of this effort include:

- Obtain Valided Vendor Supplied RTS
- Maintain Stable RTS Interface
- Modify Internals to gain Required Performance
- Address (Re) Validation issues

CONCLUSION

Although several of the issues that face developers of real-time ECS applications in Ada are design issues or primarily resolved through education, training and good programming technique, many issues remain that pose risk to the development of real-time systems in Ada.

We have identified a number of key risk areas and issues for real-time ECS applications and have explored these issues, and solutions, within the context of a specific Ada language application. With respect to the issues that were successfully addressed within the scope of this case study, the following conclusions can be made.

Many issues of concern exist due to the immaturity and quality of Ada language implementations and uncertainties regarding performance. The performance of the code generated by early compilers may be poor and may result in poor system performance. However, as Ada language systems mature and currently available optimizing technology is employed, large runtime overhead with respect to memory utilization and execution speed should certainly become less of an issue. This is in fact the case with some of the Ada language systems currently under development.

Current investigations with a variety of differing compiler systems and runtime environments for 16 and 32 bit embedded targets have revealed that kernel runtime systems currently exist that appear to be providing the minimal, configurable support necessary to accommodate Ada language features in a timely and efficient manner. Standardized kernel runtime support on the order of 2K provided by minimal system service interfaces is currently available (e.g. VRTX) and can be targeted and utilized efficiently by Ada compiler systems for a variety of embedded targets.

Problems remain with the non-support among many Ada implementations of certain real-time features of the Ada language. A case in point is the vectoring of interrupts to task entries via the Ada representation specification. This continues to be a concern to the real-time applications community although it is somewhat localized to the mainframe environment. Additional problems are rooted in the language specification itself (MIL STD 1815A) which fails to provide certain features desirable in typical real-time systems.

While alternatives exist, this lack of certain explicit language primitives poses unique problems for many types of real-time applications. Specifically, the lack of explicit language primitives to allow dynamic "disconnection" and "connection" to interrupts without the termination or creation of a program unit (task) and the inability to utilize dynamic task prioritization are of major concern to ECS developers. Furthermore, the lack of precision in the specification of exact delays as well as the lack of alternatives or ability to time-out during initiated rendezvous' may be an impediment to the development of efficient, reliable real-time systems in Ada.

There is a continuing need for a clear, concise design methodology for real-time embedded Ada applications that includes a criteria for the identification of concurrency and a graphic means of depicting concurrent relationships with timing and synchronization information at any given point in the system. While helpful, the hybrid method utilized during this case study falls short of fulfilling such a broad requirement.

We are currently continuing our real-time investigations to evaluate the effectiveness of Ada language systems for real-time embedded applications within realistic host and target environments. This work is being carried out with a focus on the 1750A and 68000 compiler and runtime environments.

The focus of our initial case study was at the applications level although an alternative was proposed for a prohibitively restrictive Ada executive that fulfilled a subset of the runtime responsibilities to support the execution of concurrent Ada programs. The current approach calls for migration to the RTS level to investigate optimization and tailoring of existing systems to allow efficient use of the Ada tasking model and other real-time features within realistic target environments. It is in this manner that we will attempt to address and seek additional information and solutions to those issues left unanswered in our preliminary Ada real-time investigations.

Options for future efficiency improvement and risk-reduction include:

- Highly Configurable Runtime Support Systems
- Standardized Runtime Support Systems
- Support in Silicon
- Custom RTS Components Libraries

ACKNOWLEDGMENTS

The authors wish to acknowledge the support and advice of the personnel at Intermetrics, Inc. in the preparation of this manuscript.

REFERENCES

1. Judge, J.F., "Ada Progress Satisfies DoD", Defense Electronics, June 1985.

2. Booch, Grady, Software Engineering with Ada, Benjamin/Cummings, Menlo Park, California, 1983.

3. Davis, R., "FDA Program Conclusions", Intermetrics Inc., Huntington Beach, California, August, 1985.

4. Laird, James D., "Implementation of an Ada Real-Time Executive: A Detailed Analysis", Intermetrics Inc., Huntington Beach, California, March, 1985.

5. Gomaa, H., "A Software Design Method for Real-Time Systems", Communications of the ACM, Vol. 27, No. 9, September 1984.

6. Temte, Mark, "Object Oriented Design and Ballistics Software", ACM Ada Letters, Vol. IV, No. 3, November/December, 1984.

7. United States Department of Defense, Reference Manual for the Ada Programming Language MIL STD 1815A, Ada Joint Program Office, March, 1983.

8. Binding, Carl, "Cheap Concurrency in C", ACM SIGPLAN NOTICES, Vol. 20, No. 9, September 1985.