Technical Report

CMU/SEI-87-TR-031
ESD-TR-87-194

Annual Technical Report for Ada Embedded Systems Testbed Project

Nelson H. Weiderman
Neal Altman
Mark Borger
Patrick Donohoe
William E. Hefley
Mark H. Klein
Stefan F. Landherr
Hans Mumm
John A. Slusarz

December 1987

Technical Report

CMU/SEI-87-TR-31
ESD-TR-87-194
December 1987

Annual Technical Report for Ada Embedded Systems Testbed Project

Nelson H. Weiderman
Neal Altman
Mark Borger
Patrick Donohoe
William E. Hefley
Mark Klein
Stefan F. Landherr
Hans Mumm
John A. Slusarz

Ada Embedded Systems Testbed Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl H. Shingler SIGNATURE ON FILE
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Annual Technical Report for Ada Embedded Systems Testbed Project

Abstract: The purpose of the Software Engineering Institute's (SEI) Ada Embedded Systems Testbed (AEST) Project is to investigate some of the critical issues in using Ada for real-time embedded applications, particularly the extent and quality of the runtime support facility provided by Ada implementations. The project's objective has been to generate new information about using Ada in real-time embedded systems. This information is in the form of benchmark test results, higher level experiment results, and lessons learned in designing and implementing real-time applications in Ada. This technical report provides an overview of the results produced in the first year of the project (through 30 September 1987). Details of these results are contained in other, referenced technical reports.

1. Introduction

1.1. Project Approach

The investigative approach of the Ada Embedded Systems Testbed (AEST) Project promotes three stages in evaluating real-time embedded systems: benchmarking; experimentation and prototyping; and designing, coding, and testing an application.

The project testbed is designed to support the process of building embedded systems applications using various Ada environments and development tools. In this way, software engineers at the Software Engineering Institute (SEI) can learn more about using Ada in real-time embedded systems and can disseminate that information to the mission-critical computer resource (MCCR) community.

Four activities were necessary to define the testbed and related efforts. First, the criteria for the testbed were refined, and detailed engineering of the laboratory was undertaken. The engineering for the VAX target and for the first non-VAX target led to the completion of the first phase of the testbed development. Both MicroVAX and MC68020 targets were implemented in the first year, and subsequent targets, such as the MIL-STD-1750A, are planned for future years.

Second, the issues and questions regarding Ada runtime environments were researched and categorized. These issues and questions will lead to criteria for runtime systems and subsequently to tests for assessing runtime systems against these criteria.

Third, the search for appropriate applications to be implemented on the testbed was undertaken. At the completion of this search, the detailed design and implementation of the application and simulated environment began. An alternative to an existing application, a single synthesized application tailored to the needs of the project, was selected for development. This application was based on the Navy's AN/WSN-5 Inertial Navigation System (INS).

Fourth, some of the Ada benchmarks available from University of Michigan and the Performance Issues Working Group of SIGAda were run under VAXELN and on the bare MC68020 target systems.

Ultimately, the AEST Project will help fulfill the SEI mission directive to improve the quality of mission-critical computer resource systems. Plans for the future are discussed in Section 7. However, the following primary information dissemination methods will continue to be used:

- direct interaction with practitioners, compiler developers, and government program office personnel
- demonstrations
- reports documenting the results of the AEST efforts
- presentation of results at symposia and conferences

1.2. Issues and Questions

The first task of the project was to provide a framework for articulating and investigating the issues and questions related to using Ada for embedded systems applications. This was accomplished in the initial months of the project and, after several iterations, resulted in a report [Weiderman 87a]. The contents of the Weiderman report have provided input for developing the criteria for the testbed and for performing the evaluation work. The report includes chapters on the embedded systems problem domain, language issues from the point of view of the MCCR application developer, language and runtime issues from the point of view of the language system implementor, and support tools for software development in the embedded environment. An annotated bibliography of references is found as an appendix to the report. It lists papers about the use of Ada in embedded systems, with particular emphasis on Ada runtime issues.

The Weiderman report also identifies the following important issues and questions related to using Ada in embedded systems:

- Are the Ada implementations for tasking, exception handling, and interrupt handling fast enough to be used in time-critical applications?
- Are Ada runtime environments small enough to be resident with embedded applications on typical military processors?
- Are the Ada timing mechanisms sufficiently precise for periodic scheduling of embedded system activities?
- Can Ada runtime environments be tailored by the user or compiler vendor so that the user does not have to pay time and space performance penalties for features of the language that are not used?
- Is it practical and desirable to provide highly optimized runtime support primitives (e.g., Ada packages containing semaphores and high-precision timing) to supplement or replace predefined language constructs?
- What is the current compile-time and runtime performance of Ada cross-compilers? What is the quality of the generated code?
- What is the current state of the tools for supporting development and testing of embedded systems? Are there intelligent linkers? Can code be put into read-only memory (ROM)?
- Should embedded systems programmers be willing to give up some of the control of the runtime environment in order to obtain the benefits of the software engineering principles gained with Ada?
- Is there a danger that the proliferation of Ada runtime environments and Ada tool sets will have as deleterious an effect on portability and reusability as the proliferation of languages had in the 1970s?

The basic issue is the ability of Ada and Ada-based software engineering tools to satisfy the conflicting requirements of generality (the ability to solve problems in a wide range of applications and over a wide range of embedded systems) and efficiency (the ability to use limited resources in an expedient manner). It is this challenge that must be addressed by all participants in the embedded system community.

1.3. Testbed Facilities

The AEST Laboratory is one of the first project computing environments at the SEI. The AEST Project established the lab to examine critical issues in using Ada for real-time embedded systems applications. The testbed has four functional components:

- host development systems
- target systems
- environment simulators
- monitoring devices

The testbed supports the basic activities involved in developing software for MCCR systems:

- compiling Ada programs
- downloading Ada programs to a target system
- simulating the environment of a target system
- monitoring Ada programs running in the target environment

Currently, the host machines are MicroVAX IIs equipped with the VAX/VMS operating system and development software. A MicroVAX II serves as the environment simulator. A Gould K115 logic analyzer provides non-intrusive testing capabilities as a monitoring device. Target processors include MicroVAX IIs under VAXELN and bare MC68020s.

In addition to these four basic components of the testbed, VT100 terminals connected to the host, target, and simulator machines provide input/output capability. The testbed components are connected to each other and to the rest of the SEI via Local Area VAXCluster and Ethernet.

Personal workstations, located in each project member's office, are connected to the lab equipment. A personal workstation is a VAXstation II equipped with the ULTRIX operating system and the X window system. The window manager gives project members a virtual terminal interface to the testbed, allowing members to carry out local activities as well as monitor testbed activities. Figure 1-1 gives a generic picture of the AEST Laboratory and its supporting workstations.

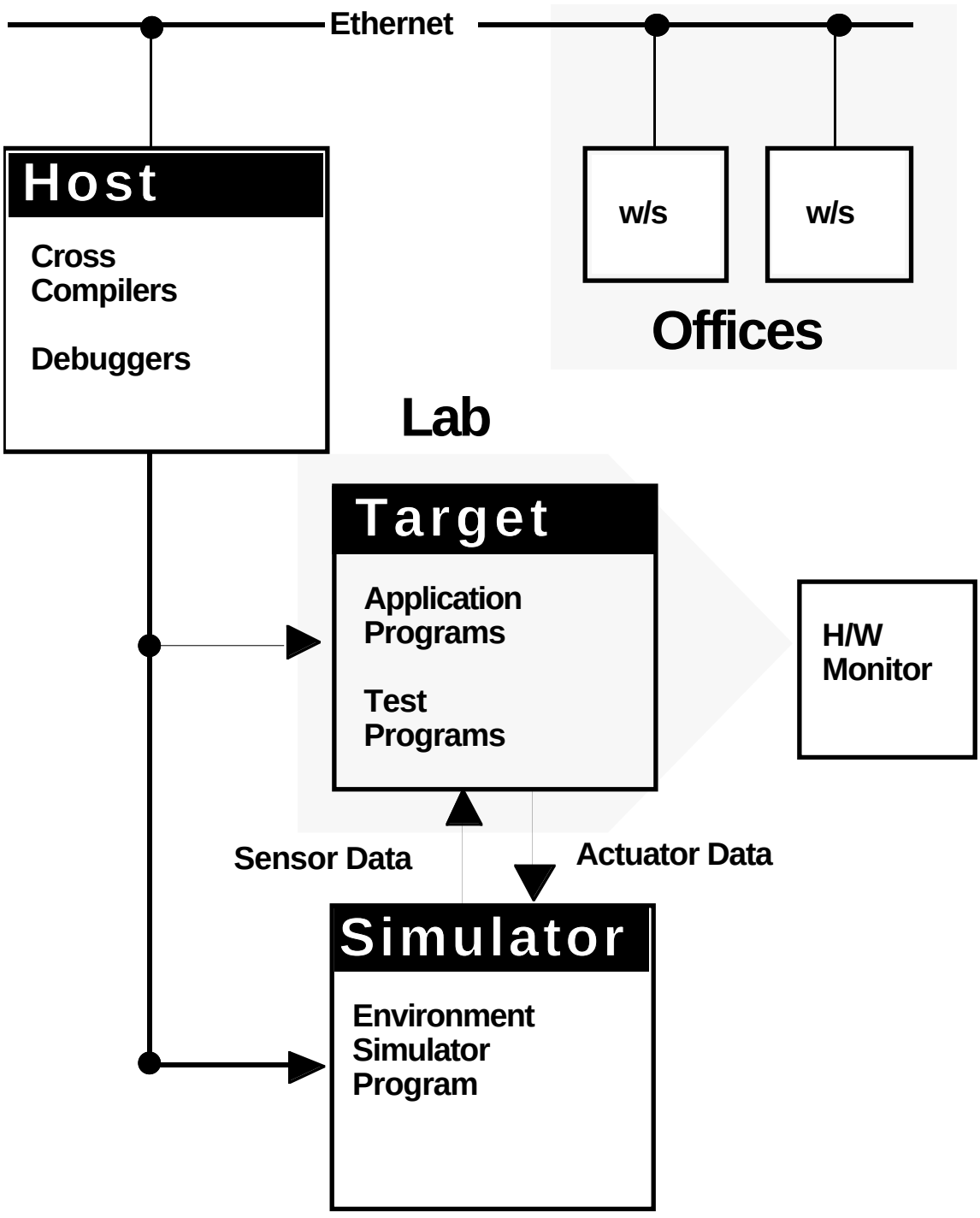

Figure 1-1: AEST Laboratory

1.4. Technology Transition

There are four audiences for technology transition work: Department of Defense (DoD) program managers, application developers, Ada compiler implementors, and Ada policy makers. The DoD program managers must have current information on tools and compilers that can be used for the demanding environment of real-time embedded systems. The application developers must know the tradeoffs involved in using various programming techniques. The Ada implementors need help in understanding the embedded system problem domain and in making implementation strategy choices in cases where the language standard allows flexibility. The Ada policy makers must know the state of the technology so that they do not mandate use of the language in certain application domains prematurely.

1.4.1. AEST Reports

Tangible deliverables in the first year were: a list of issues for using Ada in real-time embedded systems; a set of criteria for constructing a testbed for embedded systems; and an assessment of several Ada compilers currently available to support implementation of real-time embedded systems. Reports completed in the first year of the project's existence include:

- **Annual Technical Report for Ada Embedded Systems Testbed Project**. Weiderman. (CMU/SEI-87-TR-31, ESD-TR-87-194)
- **Criteria for Constructing and Using an Ada Embedded Systems Testbed**. Weiderman. (CMU/SEI-87-TR-30, ESD-TR-87-193)
- **A Survey of Real-Time Performance Benchmarks for the Ada Programming Language**. Donohoe. (CMU/SEI-87-TR-28, ESD-TR-87-191)
- **Ada Performance Benchmarks on the MicroVAX II**. Donohoe. (CMU/SEI-87-TR-27, ESD-TR-87-190)
- **Ada Performance Benchmarks on the Motorola 68020**. Donohoe. (CMU/SEI-87-TR-40, ESD-TR-87-203)
- **Timing Variations in Dual Loop Benchmarks**. Altman. (CMU/SEI-87-TR-21, ESD-TR-87-172)
- **Factors Causing Unexpected Variations in Ada Benchmarks**. Altman. (CMU/SEI-87-TR-22, ESD-TR-87-173)
- **Ada for Embedded Systems: Issues and Questions**. Weiderman et al. (CMU/SEI-87-TR-26, ESD-TR-87-189)
- **The Use of Representation Clauses and Implementation-Dependent Features in Ada: I. Overview**. Meyers and Cappellini. (CMU/SEI-87-14, ESD-TR-87-115)
- **The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIA. Evaluation Questions**. Meyers and Cappellini. (CMU/SEI-87-15, ESD/TR-87-116)
- **The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIB. Experimental Procedures**. Meyers and Cappellini. (CMU/SEI-87-TR-18, ESD-TR-87-126)
- **The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIIA. Qualitative Results for VAX Ada**. Meyers and Cappellini. (CMU/SEI-87-TR-17, ESD-TR-87-118)
- **The Use of Representation Clauses and Implementation-Dependent Features in Ada: IVA. Qualitative Results for Ada/M(44), Version 1.6**. Meyers and Cappellini. (CMU/SEI-87-TR-19, ESD-TR-87-170)

- **VAXELN Experimentation: Programming a Real-Time Clock and Interrupt Handling Using VAXELN Ada 1.1**. Borger. (CMU/SEI-87-TR-29, ESD-TR-87-192)
- **VAXELN Experimentation: Programming a Real-Time Periodic Task Dispatcher Using VAXELN Ada 1.1**. Borger. (CMU/SEI-87-TR-32, ESD-TR-87-195)
- **Inertial Navigation System Simulator: Behavioral Specification**. Landherr and Klein. (CMU/SEI-87-TR-33, ESD-TR-87-196)
- **Inertial Navigation System Simulator Program: Top-Level Design**. Klein and Landherr. (CMU/SEI-87-TR-34, ESD-TR-87-197)
- **System Specification Document for an Inertial Navigation System**. Meyers. (To be published.)
- **Functional Performance Specification for an External Computer to Interface to an Inertial Navigation System**. Meyers. (To be published.)
- **Functional Performance Specification for an Inertial Navigation System**. Meyers. (To be published.)

1.4.2. Major Meetings and Presentations

AEST personnel have attended or made presentations at numerous workshops, conferences, and meetings. These include:

- **Future APSE Workshop 1986, Saratoga Springs, NY, 9-12 September 1986**
  Mark W. Borger chaired the Evaluation and Validation Working Group at this workshop.
- **ARTEWG Meeting, Seattle, WA, 15-17 September 1986**
  Patrick Donohoe attended the Ada Runtime Environments Working Group (ARTEWG) meeting held in Seattle.
- **Ada Expo '86, Charleston, WV, 19-21 November 1986**
  Nelson H. Weiderman, Stefan F. Landherr, and Patrick Donohoe attended the Ada Expo '86 conference. Patrick Donohoe also attended the ARTEWG meeting.
- **Evaluation and Validation Team Meeting, San Diego, CA, 3-5 December 1986**
  The thirteenth meeting of the AJPO-sponsored Evaluation and Validation Team meeting was held at the General Dynamics Kearny Mesa Facility in San Diego, California. Nelson H. Weiderman participated in the Requirements Working Group.
- **ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments, Palo Alto, CA, 9-11 December 1986**
  Nelson H. Weiderman presented a paper entitled "A Methodology for Evaluating Environments." Authors of the paper were Weiderman, A. Nico Haberman, Mark W. Borger, and Mark H. Klein.
- **ACM SIGAda/AdaJUG Joint Meeting, Hollywood, FL, 12-16 January 1987**
  Mark W. Borger and John Slusarz attended this joint meeting hosted by Gould Computer Science Division. Mark W. Borger presented future APSE evaluation and validation results in the general session and presented EAE lessons learned in an APSE Builders Working Group. Mark W. Borger also attended the Ada Runtime Environment Working Group meeting.
- **Digital Equipment Corporation's VAXELN Seminar, Boston, MA, 20-22 January 1987**
  Neal A. Altman attended the DEC Seminar "VAXELN Real-Time Applications" for training in the VAXELN executive.
- **Naval Underwater Systems Center, Newport, RI, 13 February 1987**
  The purpose of this trip was to exchange information about using Ada in real-time embedded systems. The Combat Systems Division is very concerned with demonstrating the benefits of using Ada in real programs. Nelson H. Weiderman briefed the AEST Project to approximately 15 to 20 people.

- **9th International Conference on Software Engineering, Monterey, CA, 30 March-2 April 1987**
  Nelson H. Weiderman attended this conference and participated in a panel session entitled "Sampling of Existing Environments."
- **Evaluation and Validation Team Meeting, Dayton, OH, 4-6 March 1987**
  Nelson H. Weiderman attended the fourteenth meeting of the AJPO-sponsored Evaluation and Validation Team held at Wright-Patterson AFB. The Requirements Working Group (REQWG) revised the Tools and Aids document, which provides advice to the AJPO on what automated and non-automated assessment technology for APSEs ought to be procured.
- **CMU Workshop on "Fundamental Issues in Distributed Real-Time Systems," Pittsburgh, PA, 31 March - 1 April 1987**
  Mark W. Borger, Stefan F. Landherr, and Mark H. Klein attended this workshop, which was sponsored by the Computer Science Department at Carnegie Mellon.
- **ACM SIGAda Performance Issues Working Group (PIWG) Workshop, Tinton Falls, NJ, 7-8 April 1987**
  Patrick Donohoe participated in this ACM workshop, hosted by Teledyne Brown Engineering, by presenting SEI work and learning of other benchmarking efforts.
- **Military Computer Conference and Exposition, MCC '87, Anaheim, CA, 5-6 May 1987**
  Neal A. Altman attended MCC '87 as part of the AEST Project to monitor the activities and concerns of the MCCR community. A secondary purpose was to track new hardware and software suitable for inclusion in the real-time laboratory.
- **Ada Europe Conference, Stockholm, Sweden, 25-28 May 1987**
  Nelson H. Weiderman and Mark W. Borger attended a seminar conducted by the Education Working Group of AdaEurope, entitled "Improving Quality by Managing Design." Mark W. Borger presented their paper, entitled "Generic Evaluation Experiments for Assessing an Ada Environment's Support of Configuration Management Activities."
- **Dansk Datamatik Center (DDC) and DDC International A/S, Lyngby, Denmark, 29 May 1987**
  On the return trip from the AdaEurope Conference in Stockholm, Nelson H. Weiderman and Mark W. Borger visited with DDC and DDC-I to exchange information about our respective ongoing projects. DDC and DDC-I provided technical exchanges of information about DDC's formal methods work and DDC-I's Ada cross-compilers.
- **Evaluation and Validation Team Meeting, Dayton, OH, 2-4 June 1987**
  Nelson H. Weiderman attended the fifteenth meeting of the AJPO-sponsored Evaluation and Validation Team meeting held at Wright-Patterson AFB. One set of working groups follows contractual efforts (CVCWG, ACECWG, and CLASSWG), and another formulates long-term directions (REQWG, SEVWG, and COORDWG). Nelson H. Weiderman is currently serving on the ACECWG and the REQWG (Requirements Working Group).
- **SEI Affiliates Symposium, Pittsburgh PA, 9-10 June 87**
  Several AEST personnel attended this symposium, including Nelson H. Weiderman, William E. Hefley, Patrick Donohoe, Mark W. Borger, John Slusarz, Stefan F. Landherr, Mark H. Klein, and Andrea L. Cappellini. Nelson H. Weiderman presented an overview of the AEST Project at the symposium. At a separate session devoted to AEST, Nelson H. Weiderman gave a more detailed overview; John Slusarz presented the Lab configuration; Andrea L. Cappellini presented information on evaluating support for representation specifications; Stefan F. Landherr presented an overview of the INS application efforts; Mark W. Borger presented our real-time experimentation framework; and Patrick Donohoe presented the efforts of the Benchmarking/Instrumentation subgroup.

- **Northrop Corporation, Embedded Software Task Force Meeting, Hawthorne, CA, 30 June 1987**
  William E. Hefley was part of a Software Engineering Institute team that traveled to the Northrop Aircraft Division Technical Center to discuss relevant SEI projects and Northrop's concerns about the Advanced Tactical Fighter (ATF) program. The Northrop attendees included three corporate vice presidents and many managers and senior staff members in organizations such as: Avionics Software, Engineering Administration, Software Engineering, Embedded Computer Development, Avionics Systems Engineering, Software Tools, and Software Design. The organizations represented were Aircraft Division, Electronics Division, Ventura Division, and Defense Systems Division (Chicago, IL). Hefley presented an overview of the Ada Embedded Systems Testbed Project to the Task Force.
- **AdaJUG Meeting, Dayton, OH, 13-15 July 1987**
  Nelson H. Weiderman and William E. Hefley attended AdaJUG in Dayton. There were numerous discussions regarding DOD-STD-2167 issues and the needs of the MIL-STD-1750 community regarding Ada real-time issues. Hefley also attended the AdaJUG Working Group Meeting for Software Cost Estimation.
- **TRW AWIS Design Review, Fairfax, VA, 20 July 1987**
  Mark W. Borger attended this review as part of an SEI contingent invited to the review.
- **Joint Integrated Avionics Working Group (JIAWG), SEI, Pittsburgh, PA, 20 July 1987**
  Clyde Chittister, Nelson H. Weiderman, William E. Hefley, and Patrick Donohoe met with personnel from the Naval Air Test Center, Patuxent River, MD, and the Naval Avionics Center, Indianapolis. These personnel are working an embedded systems benchmarking and form, fit, and functionality testing for the Joint Integrated Avionics Working Group (JIAWG).
- **INS Application Design Review, Naval Surface Weapons Center, Dahlgren, VA, 21-22 July 1987**
  The review of the INS application design was attended by more than a dozen people each day. Larry Druffel presented an overview of the Software Engineering Institute, and Nelson H. Weiderman presented an overview of the AEST Project. Craig Meyers, former Navy affiliate, presented the motivation for doing the INS application, related it to the DoD, and related the AEST Project requirements to the real system (AN/WSN-5). Stefan F. Landherr presented an overview of the INS simulator application design and the user interface design. Mark H. Klein presented the communications design. Mark W. Borger presented the executive design. There were separate NSWC-SEI management discussion sessions, which were attended by Larry Druffel, William E. Hefley, and Major Dan Burton of the Software Engineering Institute Joint Program Office. NSWC provided a tour of the AEGIS Computer Center and a demonstration of the MicroADDS post deployment software support (PDSS) system.
- **ACM Sun Belt SIGAda Meeting, Wichita, Kansas, 19-20 August 1987**
  Mark H. Klein presented an overview of the AEST Project at Sun Belt SIGAda. The theme of this meeting was the use of Ada for real-time applications. In addition, James Tomayko arranged for Klein to present this overview to the Languages and Software group at the Boeing Military Aircraft Company (BMAC) during their monthly Technology Seminar.
- **ACM Summer '87 SIGAda, Seattle, WA, 26-27 August 1987**
  Patrick Donohoe and John Slusarz attended this SIGAda meeting for the project. Donohoe presented an overview of the AEST benchmarking effort at the PIWG meeting and provided PIWG with the latest AEST benchmark results. Donohoe also attended the ARTEWG meeting. Donohoe and Slusarz also collected information on 1750A hardware and cross-compilers and talked to vendors about current AEST cross-compiler problems.

- **Evaluation and Validation Team Meeting, Dayton, OH, 2-4 September 1987**
  Nelson H. Weiderman attended the sixteenth meeting of the AJPO-sponsored Evaluation and Validation Team meeting at Wright-Patterson AFB.
- **Armament Division, Air Force Systems Command, Eglin AFB, FL, 15 September 1987**
  Nelson H. Weiderman traveled to Eglin AFB to discuss the AEST efforts with the technical staff at the Armament Division.
- **Advanced Real-Time (ART) Project, Computer Science Department, Carnegie Mellon University, Periodic meetings throughout the year**
  Project members met with researchers John Lehoczky, Liu Sha, Hide Tokuda, and Dennis Cornhill to discuss collaborative research.

1.4.3. Government Affiliates

Government affiliates are able to transition the project's work to their home organizations, as well as adding the DoD's perspective to many issues while at the SEI. Currently, the only government affiliate on the AEST Project is Hans Mumm. Mr. Mumm is with the Naval Ocean Systems Center:

Software Engineering Technology Branch
Information Systems Division
Command and Control Department
Naval Ocean Systems Center (NOSC)
San Diego, CA 92152-5000

Past affiliates include the following individuals:

Mr. Stefan F. Landherr
Combat Systems Integration Group
Combat Systems Division
Weapons Systems Research Laboratory
Defense Science and Technology Organization
Australian Department of Defense
G.P.O. Box 2151
Adelaide SA 5001
Australia

Ms. Andrea Cappellini
US Army CECOM
Center for C^3 Systems
ATTN: AMSEL-RD-C3-IA (Cappellini)
Ft. Monmouth, NJ 07703

Dr. B. Craig Meyers
Information and Control Technology Branch
Combat Systems Department (Code N35)
Naval Surface Weapons Center
Dahlgren, VA 22448

1.5. Background

In the last two years, Ada compilers have reached a state of maturity that justifies their use in production applications. Ada software development environments, however, are lagging somewhat behind compilers in maturity and sophistication. Many of them lack tools and interfaces we have come to expect in modern software development environments, but they are still quite usable. In contrast to Ada compiler and environment technology, the ability to use Ada in real-time embedded systems is just now being explored. There are several vendors whose compilers generate code for embedded systems targets such as the Motorola MC68000 and Intel iAPX86 microprocessor families, but the state of embedded system support tools is uncertain. Since Ada was originally designed for embedded system applications, it is critically important to evaluate the readiness of the tools and techniques for developing and testing software for these applications.

Software for embedded systems differs from that for commercial data processing and most scientific data processing. It is sometimes written in assembly language, and it is sometimes written by knowledgeable hardware engineers rather than software engineers. Embedded systems programmers have to deal with odd hardware restrictions and constraints. The programs must be particularly efficient to be able to respond to real-time inputs from a variety of sensors. If Ada is to be successful in the embedded systems area, its performance and the performance of particular Ada implementations for target machines need to be demonstrated.

1.6. Purpose and Objectives

The Ada Embedded Systems Testbed (AEST) Project was initiated in October 1986 and has been continued by the SEI into 1987 and 1988. The purpose of the project is to investigate critical issues in using Ada for real-time embedded applications, particularly the extent and quality of the runtime support facility provided by Ada implementations. The Ada runtime is an execution environment that provides services such as process management, storage management, and exception handling for supporting the execution of Ada programs. In the past, these services were provided either by the application programmer or by a small real-time executive.

The primary goal of the Ada Embedded Systems Testbed Project is to develop a solid, in-house support base of hardware, software, and personnel that permits the investigation of a wide variety of issues related to software development for real-time embedded systems. The SEI support base will make it possible to assess the readiness of the Ada language and Ada tools to develop embedded systems. It will also make it possible for us to provide advice to contractors on how to circumvent problems, advice to vendors on what tools and features need to be provided, and advice to the DoD on what is possible with currently available commercial products.

One project objective is to collect, classify, track, and disseminate information about using Ada in real-time embedded systems. The issues that we are investigating are more oriented toward runtime and implementation issues than language issues. The AEST efforts have, however, been aware of language issues.

A second objective is to create and expand a general testbed for experimentation. The major deliverable of the project is the testbed itself. The testbed must accommodate different target processors, different compilers, and different tool sets. It should be flexible, reconfigurable, and evolvable so that a wide variety of experiments can be conducted using the facility. There should be both hardware and software measurement techniques so that performance data can be independently verified and collected in a non-intrusive manner. Finally, the testbed should provide a vehicle for embedded software projects at the SEI by installing, evaluating, using, and demonstrating state-of-the-practice hardware and Ada-based software applied to embedded systems development. This will be a major facility of the SEI, which could outlive the project and provide a capability to test new hardware and software.

A third objective of the project is to generate new information about using Ada in real-time embedded systems. This information should be in the form of benchmark test results, higher level experiment results, and lessons learned in designing and implementing real applications in Ada.

1.7. Contents of this Report

1.7.1. AEST Project Activities

Section 1 of this report provided a summary of the project activities during its first year. It described the project approach for investigating the use of Ada in embedded systems. The section then summarized the major issues and questions raised when Ada is used in this environment and described the generic testbed configuration, the experimentation undertaken, and the mechanisms for transitioning the results of the effort. Two SEI technical reports [Weiderman 87a, Weiderman 87b], respectively, describe the issues and questions and the criteria for testbed construction and use.

1.7.2. Testbed Environment

Section 2 describes the testbed environment, including its hardware, software, and test equipment. A description of each of the cross-compilers installed during the first year's effort is provided along with some of the experiences gained during the installations.

1.7.3. Benchmarking and Instrumentation

The benchmarking and instrumentation efforts, described in Section 3, focus on obtaining quantitative data on the performance of Ada on various target machines. This section describes evaluation of existing benchmarks, such as the University of Michigan and PIWG suites, as well as some of the problems encountered. A summary of key benchmark results completes this section. Details of this work are contained in SEI technical reports [Altman 87a, Altman 87b, Donohoe 87a, Donohoe 87b, Donohoe 87c].

1.7.4. Real-Time Experimentation

Section 4 describes the project's real-time experimentation. In these experiments, we assume the application developer's viewpoint and investigate the prospects of successfully implementing a particular embedded system function using a given Ada cross-compiler and pursuing alternative Ada solutions to implementing that function. The alternatives are intended to show the application developer

some implementation choices that may aid in solving implementation dilemmas. Experiments were pursued in three areas:

- internal data representation
- low-level input/output and device driver/interfaces
- periodic task scheduling

The results of each of these experiments is summarized in this section. The details of this work are contained in SEI technical reports [Borger 87a, Borger 87b, MeyersI 87, MeyersIIA 87, MeyersIIB 87, MeyersIIIA 87, MeyersIVA 87].

1.7.5. Application Development

Section 5 describes the development of an application, the third component in the project's study of Ada in the real-time embedded arena. The design and implementation of this application is an experimental effort. It provides a context for using information and results generated by the benchmarking and experimentation efforts of the project, and it ensures that our context for studying Ada closely approximates the context in which Ada will be used.

This section describes the AEST Project's efforts in building a subset of an actual real-time embedded application in Ada. The Inertial Navigation System simulator preserves the real-time properties of an actual onboard inertial navigation system while greatly simplifying those aspects of the system that require a high degree of application domain knowledge. The main goal of our effort has been to record lessons learned concerning the use of Ada in the real-time embedded arena.

The results of this application development as of 30 September 1987 are given in this section. Details of the work are given in SEI technical reports [INS Behavioral Specification 87, INS-tldd 87, Meyers 87, Meyers-EC 87, Meyers-INS 87].

1.7.6. Summary and Future Directions

The summary section, Section 6, concludes the annual report by providing a look at the future directions for the Ada Embedded Systems Testbed Project and discussing how these efforts will support new research directions at the SEI.

2. Testbed Environment

2.1. Background

A primary objective of the AEST Project is to create and expand a general testbed for experimentation. The testbed must accommodate different target processors, different compilers, and different tool sets. It must be flexible, reconfigurable, and evolvable. There must be hardware and software measurement techniques so that performance data can be independently verified and collected in a non-intrusive manner. Detailed criteria for constructing and using the testbed were developed by the project staff [Weiderman 87b]. This chapter provides an overview of the testbed environment and the cross-compiler installation procedure.

2.2. Hardware Configuration

The testbed host machine inventory consists of four DEC MicroVAX IIs, all with standard configurations that include 8 to 12 megabytes of memory and Ethernet connections. These hosts are connected to a common file server using VAXCluster software. The host operating system is VAX/VMS.

There are two types of target machines in the testbed. The first is a DEC MicroVAX II in a "bare" configuration. These targets are designed to be VAXELN targets and are connected to the Ethernet, which is the downloading path.

The second target type is the MVME133A single-board computer using a Motorola MC68020 as the processor and a MC68881 floating point coprocessor. The processor runs at 12.5 MHz with one wait state. These boards contain 1 MByte of random-access memory (RAM), a debugger/monitor based on programmable read-only memory (PROM), 3 serial ports, and 2 timers. They are located in a VME bus chassis which provides power and cooling. The memory on each card is dual ported, which will allow multiprocessing experiments in the future.

All communication to the testbed from remote locations is provided by the Ethernet system that is global to the SEI. Members of the project staff typically use personal workstations running DEC's ULTRIX operating system with the X window system to open windows into the specific host and target computers. All required functions on hosts and targets are available with this method. Thus, project members conduct compilation, downloading, and target execution from offices distributed around the SEI and even from terminals in their homes.

2.3. Test Environment

2.3.1. Hardware Test Devices

The AEST Project acquired a Gould K115 logic analyzer to perform precise timing measurements on Ada code. This logic analyzer is capable of monitoring 64 data lines for 1,024 sample times. It is designed to be a device for testing hardware, but it can be configured for use with a MC68020 so that data, address, and control line transitions are captured. This mode is useful for debugging software

since a symbolic disassembler is provided. However, this mode is not useful for timing because the interface removes timing information during data capture. The logic analyzer is not a perfect tool for software debugging as it is not able to capture data from complicated software routines.

2.3.2. Software Test Devices

The Motorola MVME133A has a monitor/debugger package located in PROM. This package allows most of the normal debugging operations such as step, examine memory, modify memory, set/clear breakpoints, and run. The monitor provides the capability to establish communications with the host computer over the serial lines and to download files.

2.4. Cross-Compiler Configurations

This section provides some insight to the problems we encountered during the installation of cross-compilers into the testbed environment. These problems can be attributed in part to our inexperience with cross-compiler installation, but other contributing factors include immature products, incomplete documentation, and in some cases unsatisfactory customer service. The unsatisfactory customer service was due partly to the vendor's inexperience with cross-compiler products and partly due to a lack of comparable target hardware at vendor sites.

2.4.1. Typical Configuration

To understand the problems one may encounter, it is useful to understand the interface between the host system and the embedded target. The cross-compiler runs on a host computer, a MicroVAX II running VMS in our case. The cross-compiler compiles the Ada code and links the Ada code into an executable image. This executable image must be transported to the target computer; that transfer requires a communications link between the host computer and the target computer. In the AEST testbed, the communications link is Ethernet or an RS-232 serial port. The cross-compiler vendor must provide two modules of communications software, one to reside on the host computer and one to reside on the target computer. The host module, residing on the host computer, presents no difficulties because the vendor is able to test the host communications software. The target software is more challenging because the vendor must provide the hooks to allow modifications to various target configurations. The users then must write their own device driver for their particular target. In most cases, vendors provide several examples of device drivers. The "best fit" driver may then be used as the starting point for the configuration of the target serial port device drivers.

After the modifications are made to the target's communication software, its executable image is downloaded to the target. For the MC68020s, this download was performed using the Motorola board-level debugger/monitor. Another method would be to burn a PROM for the target single-board computer.

Debugging the target serial driver can be a nontrivial problem. The only tools available in the AEST lab were the Motorola debugger and a Gould logic analyzer. Each provided some help, but each had its own problems. The Gould was unable to capture sufficient information in complicated looping situations. The debugger frequently was disabled by the target resident code. Developers with access to a microprocessor development station would not observe these problems.

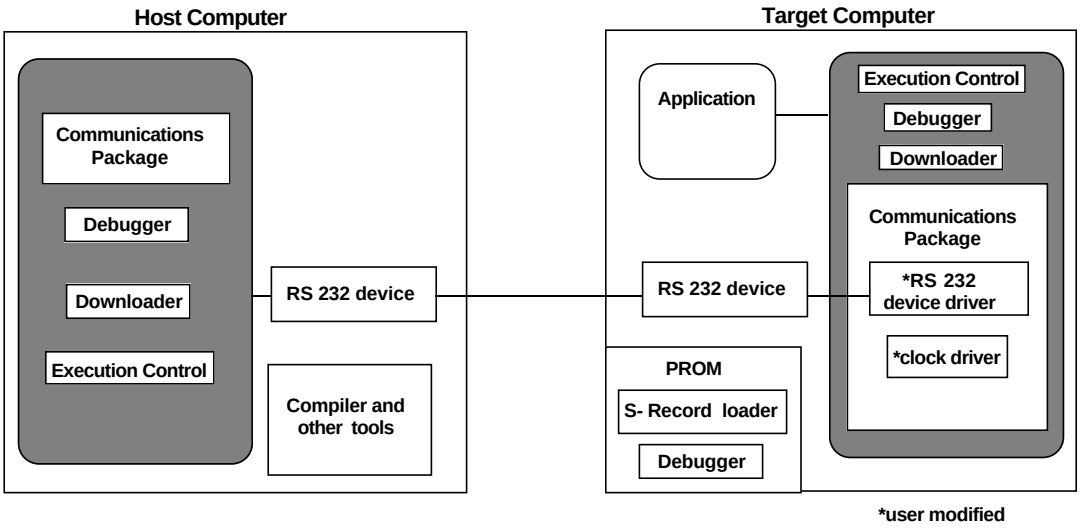

Figure 2-1: Typical Host Computer/Target Computer Configuration

Some vendors provided host-based test code and routines to verify the proper operation of the target resident code. This was useful since the tests also provided "symptoms" that could be reported when calling customer support.

Each target resident package also contained sections of code that related to the target computer's clock or timing device. Again, these routines were modified to fit the target hardware. The most difficult aspect of the modification was understanding the details of the hardware clock. The code modifications were not difficult.

2.4.2. Vendor: Digital Equipment Corporation

Product : VAXELN Ada, Version 1.1 with VAXELN, Version 2.3
Host Computer : MicroVAX II
Target Computer : MicroVAX II

Installation

The VAX to VAX configuration was used as the first target because of our experience with DEC equipment and DEC Ada compilers on a previous project. The communications link was Ethernet rather than RS-232. The installation did not present any special problems, in part because all the components were supplied by a single vendor. Since this is not a typical cross-compilation system, the installation is not fully described.

2.4.3. Vendor: Systems Designers plc.

Product : SD Ada-Plus, Version 2B.01
Host Computer : MicroVAX II
Target Computer : Motorola MVME133A

Installation

Package arrived on February 4, 1987.
Operational on March 24.
(The target system was not available until March 10th.)

The person installing this product had problems with the installation procedure. His comments indicated that the procedure could be simplified; he believed that without his level of expertise in VMS, he would not have been able to complete the installation. The rest of the installation and checkout did not present any major difficulty. No changes were made to the code since our target was the same target System Designers (SD) used in their development.

The documentation provided with the SD product left much to de desired. The layout was poor; information was hard to find; and there were few examples. In addition, the command syntax and semantics were difficult to understand.

2.4.4. Vendor: TeleSoft

Product : TeleGen2, Version 3.13
Host Computer : MicroVAX II
Target Computer : Motorola MVME133A

Installation

Package arrived on June 19, 1987.
Installed on host system on July 9.
Operational on July 15.

The TeleSoft product is in the class of products that modify the VBR (vector base register, which points to the vector table) and other individual interrupt vectors, which disable the PROM-based debugger on the target board. When contacted for support, TeleSoft was unable to provide an immediate answer, but they did have the configuration file for the MVME133A. The file was mailed and installed, but the distribution tape should have included all the target-specific files that TeleSoft had available. The time spent making modifications to target files could have been saved. Compiler vendors should also strive to acquire a collection of target types to broaden their experience.

The cross-debugger provided with the TeleSoft package is still not operational because of a communications problem with the host. The communications package seems to raise the same exception for many different problems instead of providing more information about what raises the exception. We have contacted customer support, but it is difficult for them to help. Some of the support personnel we have talked to do not have the direct experience required to resolve these types of problems. One difficulty seems to be that some of them have never seen a target machine, nor do they have direct access to a target machine. Typically, they talk to the designers and call back. In this case the issue is still open.

2.4.5. Vendor: VERDIX

Product : VADS Ada, Version 5.41(h)
Host Computer : MicroVAX II
Target Computer : Motorola MVME133A

Installation

Package I arrived on March 10, 1987. After we had worked on this release for six weeks, VERDIX informed us that it was a prerelease version and that a new version was coming out. Package II arrived on May 25, 1987, and was installed on June 16, 1987. Our system manager had to make several changes before the install procedure worked.

Our main problem seemed to be a dependency problem where recompilation of some units was required. The modified serial routines would not link. With the new release, the source files that were called out as obsolete were not included. In addition, several of the units were listed as "source error" in a library report.

Over the next ten days of discussion with VERDIX customer support, we learned the following: the source files were not available with this release; "source error" refers to lack of source information in the library; the target portion must be built using the library for a MC68020 without the floating point coprocessor; and the body of the main unit must be compiled in the library that contains the specification of the main unit. All these items were corrected, but the link dependency problem persisted.

At that point, VERDIX notified us that the version we had was defective and a new version was required. Package III arrived on July 29, 1987. This package appeared to have the same problems as the last package. After further discussion with VERDIX, we both learned for the second time that the body must go with the specification for the main unit. In the process, we also discovered that the VMS file dates and times are not used by the VERDIX system; it maintains it own timestamps. Fixing this problem resulted in a successful build.

During the debugging of this code, a conflict was discovered. The installation had proceeded to the point where it was possible to download code but not to execute. The conflict was traced to the use of trap #15 by both the PROM-based monitor and the application. Trap #15 had been disabled to allow use of the PROM-based debugger. Restoring trap 15 to the VERDIX environment allowed applications to run successfully at the cost of the board-level debugger. The system finally became operational on August 31, 1987.

We encountered the same specification/body library problem with the timer modifications. The body needed to be in the same library as the specification. The library tools were no help in resolving this problem. No error messages were produced, and the documentation did not mention the problem.

Another problem was discovered with the timer and the timer interrupt vector. The first application run would set the timer vector to its handler. This program would run; but when a new application was downloaded, a system crash would result. This problem occurred because the timer interrupt fired after the new code was downloaded (with a new address of a handler) but before the timer interrupt vector was modified. The problem has not yet been resolved, but it appears that the downloader should disable timer interrupt and the application/environment re-enable timer interrupts at the appropriate time. The timer became operational on September 3, 1987.

3. Benchmarking and Instrumentation

3.1. Objectives and Approach

The benchmarking/instrumentation subgroup was formed with the goals of obtaining data on the real-time performance of Ada on several different target machines and providing a firm practical and theoretical foundation on which other AEST activities can draw.

The objectives of the subgroup are:

- Collect and run available Ada benchmark programs from a variety of sources on a variety of targets.
- Identify gaps in the coverage and fill them with new test programs.
- Review the measurement techniques used and provide new ones if necessary.
- Verify software timings by inspection and by using specialized test instruments.

For practical purposes, the group decided to use existing Ada benchmarks initially and write additional AEST benchmarks if needed. The three major suites of Ada benchmarks known to the AEST Project at its inception were the University of Michigan benchmarks [Clapp 86], the ACM Performance Issues Working Group (PIWG) benchmarks,[1] and the prototype Ada Compiler Evaluation Capability (ACEC) [Hook 85]. An early AEST report [Donohoe 87a] assessed the applicability of these suites to the AEST Project; and the group decided to focus on the Michigan and PIWG benchmarks initially. Since then, however, Boeing Military Airplane Company has been awarded a contract by the Evaluation and Validation Team of the Ada Joint Program Office to produce a full set of ACEC tests; the SEI will participate in testing the ACEC suite.

Automated procedures were developed to build, download, and run the benchmarks on the testbed's VAXELN-based MicroVAX II. Based on this experience, modified techniques for handling selected benchmarks were developed for the Motorola MC68020. A brief discussion of the problems encountered and some key results appear in later sections of this report. The full results of the VAXELN MicroVAX II test runs are reported in [Donohoe 87b]; MC68020 results for the SD Ada-Plus, the TeleSoft TeleGen2, and the VERDIX cross-compilers are reported in [Donohoe 87c].

3.2. Existing Ada Benchmarks

The University of Michigan Ada benchmarks and the PIWG benchmarks concentrate on techniques for measuring the performance of individual features of the Ada programming language. They do not address such issues as efficiency or size of generated object code. Also, they measure language features in isolation, not interactions between features (e.g., tasking combined with exception handling). The two suites of benchmarks are described in a separate AEST report [Donohoe 87a]. Briefly, they measure such features as task rendezvous, task elaboration, activation and termination,

[1] The benchmarks came from the PIWG distribution tape known as TAPE_8_31_86. The name, address, and telephone number of the current chairperson of the PIWG can be found in *Ada Letters*, a bimonthly publication of SIGAda, the ACM Special Interest Group on Ada.

exception handling, dynamic storage allocation, subprogram calling overhead, and CALENDAR.CLOCK function calling overhead. The PIWG suite also contains Ada versions of the composite synthetic benchmarks Whetstone [Curnow 76], and Dhrystone [Weicker 84], as well as tests to measure compiler efficiency. The measurement techniques of these benchmark suites and some of the problems encountered when running the tests are described in the next section.

3.3. Benchmarking Techniques and Problems

To isolate a specific feature for execution time measurement, a typical Michigan or PIWG benchmark program executes a control loop and a test loop, the loops differing only by the feature to be measured. Each loop is run for the same number of iterations, usually 10,000. Time readings are taken at the beginning and end of both loops by calling the Ada CALENDAR.CLOCK function. Theoretically, the difference in execution times of the two loops divided by the number of iterations is the execution speed of the feature of interest. The technique of averaging over a large number of loop iterations is used to overcome the limitation of coarse clock resolution by ensuring that the duration of the loops is long enough for the clock calls to return different values.[2] The dual loop scheme also incorporates techniques to prevent the loop structures from being optimized away by a smart compiler.

The major problem encountered when running the benchmarks was the appearance of negative times in the output of some of the Michigan dynamic allocation tests. An investigation of the problem challenged the fundamental assumption of the benchmarks: that textually equivalent loops execute in similar amounts of time. When the language feature being measured was removed from the test loop, the difference between its execution speed and that of the control loop did not zero out as expected. The details of the investigation appear in [Altman 87a]. The principal reasons for the negative times, in the case of the MicroVAX VAXELN target, were the alignment of the loops with respect to page boundaries (loops spanning page boundaries took longer to execute) and the placement of subprograms within packages (the first subprogram in a package always contained one less machine code instruction than subsequent subprograms). Tracking down and identifying the source of these problems proved to be a time-consuming task, complicated by the fact that VAXELN link maps display virtual rather than real addresses.

The dual loop problem prompted a more general examination of the nature of benchmarking and the effects of particular hardware features (e.g., paging, cache memory, alignment) and software features (e.g., optimization, memory allocation, garbage collection). A discussion of these and other topics appears in a separate report [Altman 87b]. To collect more data points for future analysis, the staff compiled as many Michigan and PIWG tests as possible and ran them using the various cross-compilers and targets of the testbed. The results are documented in [Donohoe 87b] and [Donohoe 87c].

[2]The clock resolution for VAXELN Ada is 10 milliseconds; for SD Ada-Plus, it is 7.8 milliseconds; for TeleGen2, it is 100 milliseconds; and for VERDIX, it is 61 microseconds. Many of the benchmarked language features executed in tens of microseconds.

Multiple runs of PIWG benchmarks compiled under TeleGen2 sometimes produced variations in the reported timing measurements. An investigation uncovered two issues: the apparent accuracy of PIWG timing measurements, and the disparity between the value of SYSTEM.TICK and the resolution of CALENDAR.CLOCK. One of the PIWG benchmark support packages, A000032.ADA, contains the body of the ITERATION package. This package is called by a benchmark program to calculate, among other things, the minimum duration for the test loop of a benchmark run. The idea is to run the benchmark for enough iterations to overcome the problem of the relatively coarse resolution of CALENDAR.CLOCK. The minimum duration is computed to be the larger of 1 second, 100 times SYSTEM.TICK, and 100 times STANDARD DURATION'SMALL. There are two problems with this scheme:

- The times reported by the benchmark programs appear to be accurate to one tenth of a microsecond; however, merely running the test for a specific minimum duration does not guarantee this degree of accuracy. If the clock resolution is 10 milliseconds, for example, and the desired accuracy is to within one microsecond, then the test should be run for 10,000 iterations. For Ada language features which execute in tens of microseconds, running for a specific duration may ensure enough iterations for accuracy to within one microsecond; this is not so for language features which take longer.
- Since SYSTEM.TICK is used in the minimum duration calculation, the implicit assumption seems to be that SYSTEM.TICK is equivalent to one tick of CALENDAR.CLOCK. This is not necessarily so. For example, for the TeleGen2 MC68020 cross-compiler, SYSTEM.TICK is 10 milliseconds but the resolution of CALENDAR.CLOCK determined by the University of Michigan calibration test is 100 milliseconds. (The TeleSoft documentation states that SYSTEM.TICK is not used by any component of the TeleGen2 compiler or runtime system.)

In general, the accuracy of the PIWG and Michigan benchmarks is to within one tick of CALENDAR.CLOCK divided by the number of iterations of the benchmark (see the "Basic Measurement Accuracy" section of the University of Michigan report [Clapp 86]). The University of Michigan benchmarks typically run for 10,000 iterations, so they are accurate to within 1 microsecond or better for VAXELN Ada (10 millisecond clock resolution), SD Ada (7.8 millisecond clock resolution), and VERDIX Ada (61 microsecond clock resolution). For TeleGen2, they are accurate to within 10 microseconds.

Other problems arose when attempts were made to verify some of the timing results using the logic analyzer. These included: identifying the beginning and end of the assembly code generated for the language feature being measured; computing the actual start and end addresses of this code by examining load maps, adding appropriate offsets, and allowing for the MC68020 word-bounding; and choosing the correct "window" to capture data. Because of these problems, only the Michigan and PIWG task rendezvous times for the SD cross-compiler have been verified using the logic analyzer; these times are within 5 percent of the times reported by the benchmark runs. More use of the logic analyzer will be made in the next phase of the AEST Project, when a PC with a data storage and analysis package will be connected to the analyzer.

The remaining problems encountered during this benchmarking effort related to the varying degrees of difficulty experienced in getting the cross-compilers installed and running, and getting the benchmarks to run on the target machines, particularly the MC68020. Also, the SD cross-compiler had its own special problem; programs compiled using the SD cross-compiler had to use a

TARGET_IO package, instead of TEXT_IO, to produce output from the target machine. This meant that all benchmark output statements had to be converted to use the routines provided by TARGET_IO. An additional problem was then discovered: the routine to print floating point numbers never produced any output. Examination of the source of TARGET_IO (which was fortunately provided with the SD product) revealed that the routine had a null body. Rather than write a floating point routine, a quick solution was to scale up the timing results—in microseconds—and use the integer output routine. This meant that times would be measured to the nearest microsecond instead of to the theoretically achievable tenth of a microsecond. For the PIWG benchmarks, resolving the shortcomings of TARGET_IO was only a matter of changing a single general-purpose output program. For the Michigan benchmarks, however, it would have meant changing each of the output routines associated with the individual tests. For this reason, not all of the Michigan benchmarks were converted for the SD cross-compiler.

3.4. Key Results

This section presents sample results for some of the more important real-time Ada language features. The results shown are for benchmarks for which the dual loop timing bias is believed to be the smallest. The VAXELN and SD Ada compilers provide optimization[3] by default; TeleGen2 and VERDIX Ada require that optimization be requested explicitly, by means of a qualifier on the compile command. The TeleGen2 and VERDIX Ada results presented here are for unoptimized runs.

In the results shown below, a "simple" rendezvous means one with no passed parameters, no select statements, and no accept statement body. For the exception handling test, the exception is user-defined and explicitly raised and handled in the same block of code. Clock overhead measures the time taken to read a CALENDAR.CLOCK value. The dynamic storage allocation result is for a 1000-integer array. All times are in microseconds, and the accuracy of a particular measurement is stated explicitly in the cases where it is not to within one microsecond.

[3]The VAXELN Ada compiler performs a number of standard optimizations, including: elimination of common sub-expressions; removal of invariant computations from loops; inline code expansion; global assignment of variables to registers; peephole optimization of instruction sequences; and elimination of dead code. No information about SD or VERDIX optimizations was provided in the documentation.

U. Michigan Simple Task Rendezvous (r_rend.ada)

MicroVAX II		
DEC VAXELN Ada	1569	
MC68020		
SD Ada-Plus	480	
TeleGen2 (not optimized)	480	+/- 10 microseconds
VERDIX (not optimized)	329	

PIWG Simple Exception Handling (E000001.ADA)

MicroVAX II		
DEC VAXELN Ada	786	+/- 6 microseconds
MC68020		
SD Ada-Plus	32	
TeleGen2 (not optimized)	437	+/- 4 microseconds
VERDIX (not optimized)	7035	

U. Michigan Clock Overhead (co_main.ada)

MicroVAX II		
DEC VAXELN Ada	84	
MC68020		
SD Ada-Plus	100	
TeleGen2 (not optimized)	350	+/- 10 microseconds
VERDIX (not optimized)	1270	

PIWG Dynamic Storage Allocation (D000001.ADA)

MicroVAX II		
DEC VAXELN Ada	38	
MC68020		
SD Ada-Plus	9	
TeleGen2 (not optimized)	12	+/- 4 microseconds
VERDIX (not optimized)	35	

The task rendezvous time shown for SD Ada-Plus has been verified to within 3 percent using the logic analyzer. No runtime checks were suppressed in any of the tests. Apart from the changes to the output routines necessitated by the SD cross-compiler, the source code of the benchmarks was not altered in any way.

3.5. Conclusions

Once the initial VAXELN benchmarking effort started to turn up problems, it became clear that merely grinding through the tests and producing lists of numbers would be of little use. Rather than being an end in themselves, the timing results obtained from the different compiler/target combinations became data points for current and future benchmarks evaluation. To date, the AEST Project has been able to explore only a few of the many factors which potentially affect the reliability of benchmark results. Our experience has led to a new appraisal of the role of benchmarking within the project and will determine the future course of action for the benchmarking subgroup. The following are some of the major conclusions reached as a result of work to date:

- It is very important to check the underlying assumptions incorporated in the benchmark design before attempting to use the benchmark. A simple example of such a check is a "calibration" routine to determine whether or not a dual loop test with textually identical loops will zero out.
- Even when few problems (or none) are encountered while running benchmarks, the results should be checked for reasonableness, especially if the times reported are different from heuristically calculated figures.
- Inspection of generated assembly code can uncover clues to puzzling results. Once problems start cropping up, knowledge of the machine's instruction set architecture and underlying hardware can prove useful.
- It is highly desirable to have more than one method of timing the tests; for example, the AEST Project has successfully used a logic analyzer, albeit to a limited extent, for benchmark timing. Efforts are also under way to construct a software test harness which uses the faster timers available on the testbed target machines. The initial test harness will use the KWV11-C real-time clock on the MicroVAX II target.
- The time taken to install cross-compilers and actually get benchmarks to run on the target can take considerably longer than one might expect. The process is highly dependent on good vendor documentation and support, and on the maturity of the cross-compilation systems themselves.

It is difficult to generalize about benchmarking results. The AEST effort has shown that results must be interpreted within the context of a particular hardware and software environment. Benchmarks which measure individual language features are not appropriate for inter-machine comparisons. More application-oriented benchmarks (for example, portions of an avionics application containing representative language feature mixes) are needed for such testing; unfortunately, much of this code is proprietary. It is hoped that the Inertial Navigation System application will provide some help in this area.

4. Real-Time Experimentation

4.1. Introduction

This chapter provides the reader with some highlights and lessons learned from the real-time experimentation that has been conducted as part of the Ada Embedded Systems Testbed (AEST) Project. This experimentation targeted some important issues and questions related to using Ada in embedded systems, such as those listed in Section 1.2.

These questions were pursued with experiments in three areas:

- internal data representation
- low-level input/output and device driver/interfaces
- periodic task scheduling

4.1.1. Background

To provide a framework for articulating and investigating the issues and questions related to using Ada for embedded systems applications, one of the AEST Project's initial tasks was to write an "issues and questions" report [Weiderman 87a]. By the completion of that report, several hundred issues and questions had been enumerated. To facilitate future work, we divided them into two categories: one for the issues and questions that would require some research or experimentation in order to resolve; the other for those that are really factual statements and/or recommendations and would not need to be formally addressed. The real-time experimentation described in this chapter addresses the issues in the first category—those which are not easily answered by using simple test programs.

In order to rank the importance of resolving the issues raised and, therefore, select experiment areas to investigate, the project staff established a set of criteria [Weiderman 87b]. This set of criteria, which guided the selection and design of the project's real-time experiments, is summarized below.

1. The experiments should support the Inertial Navigation System (INS) application development.
    - representation clauses: bit manipulations; converting machine representations of values into communications protocol format
    - tasking with priorities
    - periodic scheduling: tasks scheduled on basis of time intervals; scheduler invoked every timer interrupt
    - interrupt handling: timer interrupts for scheduling; I/O (e.g., communication between external computer and INS) is interrupt driven
    - use of math library: trig functions, matrix manipulations
    - buffering mechanisms: shared storage used for intertask communication

2. The experiments should provide relevant information to real-time application developers.
    - low-level I/O: interrupt handling
    - concurrent control: multitasking capabilities; determine scheduling strategy employed
    - application schedulers: cyclic; event/data driven; periodic
    - time measurements: interrupt latency; context switch time; precision (overhead) of delay

- time management: package CALENDAR; clock resolution; SYSTEM.TICK; DURATION'SMALL
- internal representation: ability to access bits
- error handling: exception handlers
- pragmas: identification of those supported; implementation semantics
- memory management: static and dynamic allocation; garbage collection
- numerical computation

3. The experiments should be feasible with respect to available resources (people, equipment).
4. The level of detail should be above that of the issues/questions and should address many aspects of an experimentation area rather than a specific issue or question.

The approach of the real-time experiments is to investigate, from a real-time application developer's viewpoint, the prospects of successfully implementing a particular embedded system function (e.g., interrupt handler) using a given Ada cross-compiler and then, where possible, to pursue alternative Ada solutions to implementing that function. The alternatives are intended to show the application developer some implementation choices that may aid in solving implementation dilemmas. Our general experimentation approach can be summarized as follows:

Step 1 Demonstrate whether or not the functionality under investigation can be implemented in Ada. The Ada solution should use the most appropriate mechanisms provided by the cross-compilation system under investigation.

Step 2 Identify additional Ada solutions for implementing the functionality.

Step 3 Implement the additional Ada solutions.

Step 4 Collect performance measurements for each Ada alternative. Analyze the compiler-generated code for each in order to identify performance bottlenecks and assess their relative cost (i.e., runtime overhead).

Step 5 Assimilate results and offer recommendations to both the Ada compiler implementor and the application developer. Provide technical information in the form of technical reports regarding how to solve the particular embedded system computing problem associated with the experiment.

4.1.2. Purpose of Experiments

The real-time experiments were conducted for the following purposes:

- Assess the feasibility of coding essential embedded system functions in Ada.
- Identify, where applicable, programming alternatives for implementing such functionality.
- Perform detailed analysis of the relative runtime costs associated with the implementation alternatives.

The results produced by the experiments are useful to a diverse audience: application developers, compiler implementors, and program managers alike. In particular, the knowledge that common computing problems in embedded systems can be solved in Ada, the comparison of alternative solutions, and the runtime performance analysis have provided some insight into the capabilities of various Ada cross-compilation systems.

4.1.3. Scope of Experimentation

The real-time experimentation that has been conducted to date has focused on three areas: internal data representation; low-level input/output and device driver/interfaces; and periodic task scheduling. It has involved two target systems (MicroVAX II/VAXELN 2.3 and a bare MC68020) and various Ada cross-compilers hosted on a VMS-based MicroVAX II. Each of the next three sections describe experiments in one of these areas. Each section contains a description of the experiment(s), an explanation of the experimental approach, a summary of empirical results, and a discussion of the lessons learned from conducting the experiment(s).

4.2. Internal Data Representation Experiment

This section describes our approach to the internal data representation experiment and summarizes results and lessons learned. In spite of the attempt to define Ada as a general-purpose language, the need to support implementation-dependent functionality has remained. Much of this support is defined in terms of representation clauses and representation-dependent features which are discussed in Chapter 13 of the *Reference Manual for the Ada Programming Language* (LRM) [U.S. Department of Defense 83]. The internal data representation experiment has been conducted to explore the use of representation clauses and implementation-dependent features of Ada and to investigate the alternative support provided by different compilers.

4.2.1. Approach

Our findings, as well as the work conducted, have been documented in a series of five reports. The initial report, "The Use of Representation Clauses and Implementation-Dependent Features in Ada: I. Overview" [MeyersI 87], presents an overview of the aspects of Ada that relate to representation clauses and implementation-dependent features. Particular emphasis is given to the use of Ada for application to packed data structures. The report is, in part, tutorial. Several examples from real-time, mission-critical systems are discussed in detail. The report also contains a brief discussion of design guidelines for the use of representation clauses and implementation-dependent features.

The second report, "The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIA. Evaluation Questions" [MeyersIIA 87], specifies a set of issues relevant to evaluating the support a given compiler provides for representation clauses and implementation-dependent features. The document lists more than fifty specific qualitative and quantitative questions about the support provided by the Ada compiler and the runtime environment. Subsequent reports then answer these questions for several compilers.

The third report, "The Use of Representation Clauses in Ada: IIB. Experimental Procedures" [MeyersIIB 87], defines and illustrates experimental procedures for the assessment of the support provided for a given compiler by representation clauses. The principal focus has been on developing a methodology applicable to such an assessment. The proposed methodology incorporates the use of program generators that automatically generate test programs and invoke analysis tools. The analysis tools include those that both collect and analyze data.

The fourth report, "The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIIA. Qualitative Results for VAX Ada" [MeyersIIIA 87], provides an assessment of the support of the representation clauses and implementation-dependent features in Ada that are provided by the VAX Ada compiler. The specific questions raised in [MeyersIIA 87] are answered.

Similarly, the fifth report, "The Use of Representation Clauses and Implementation-Dependent Features in Ada: VA. Qualitative Results for Ada/M(44)" [MeyersIVA 87], assesses the Ada/M(44) compiler.

4.2.2. Results

The results presented below summarize by functional category the support provided by the VAX Ada and Ada/M(44) compilers. The support is given a subjective rating, and explanatory information is included where relevant. More complete information is contained in reports [MeyersIIIA 87] and [MeyersIVA 87].

1. Pragma OPTIMIZE:
    a. VAX Ada - Full support
    b. Ada/M(44) - Full support

2. Data Types:
    a. VAX Ada - Full support. VAX Ada provides additional representations of integer and floating point types beyond the scope of the language. Note, however, that all fixed point types are represented using a length of 32 bits.
    b. Ada/M(44) - Supported with minor limitations. The lack of support for floating point types with precision greater than six digits may cause problems for some applications.

3. Pragma PACK:
    a. VAX Ada - Full support. Note that fixed and floating point types within records cannot be packed on arbitrary bit boundaries.
    b. Ada/M(44) - Full support. No documented restrictions.

4. Length Clauses:
    a. VAX Ada - Supported with minor limitations. Fixed point types are always the same size. This could present problems for certain applications.
    b. Ada/M(44) - Supported with minor limitations. Fixed point types are always the same size. This could present problems for certain applications.

5. Enumeration Representation Clauses:
    a. VAX Ada - Full Support.
    b. Ada/M(44) - Full Support.

6. Record Representation Clauses:
    a. VAX Ada - Supported with minor limitations. Where a record contains a component of a fixed point type, restrictions on the size of the component may be a problem due to the limitations imposed by the compiler.

b. Ada/M(44) - Supported with minor limitations. Where a record contains a component of a fixed point type, restrictions on the size of the component may be a problem due to the limitations imposed by the compiler.

7. Address Clauses:

   a. VAX Ada - Supported with major limitations. For VAX Ada, the simple name in the address clause may only be the name of a variable. A simple name that names constants, subprograms, packages, tasks, or single entries is not supported.
   b. Ada/M(44) - Supported with major limitations. The simple name in the address clause may only be the name for a single task entry. Ada/M(44) does not support the other allowable names.

8. Data Conversion and Assignment:

   a. VAX Ada - Full support. The result of UNCHECKED_CONVERSION when source and target type differ in length may cause problems for some applications.
   b. Ada/M(44) - Full support.

9. Representation Attributes:

   a. VAX Ada - Full support. VAX Ada also provides two additional representation attributes which may be of benefit. One allows the user to determine the number of machine bits to be allocated for variables of a type or subtype. The other yields the bit offset of the object within a storage unit.
   b. Ada/M(44) - Full support. Ada/M(44) provides an additional attribute which allows the user to access physical addresses and may be helpful for some systems.

10. Pragma INTERFACE

    a. VAX Ada - Full support for interface to VAX-supported languages such as assembler, FORTRAN, Pascal, and C. While the ability to interface with other languages is supported, it may involve considerable effort to achieve the desired result. Note that VAX Ada allows users to specify the details of parameter passing.
    b. Ada/M(44) - Supported with major limitations. Only subprograms written in assembler are supported.

11. Support Facilities

    a. VAX Ada - Full support. In general the documentation and debugger were found to be excellent. Note, however, that the release notes are difficult to use.
    b. Ada/M(44) - Supported with major limitations. No debugger is available for the AN/UYK-44 target. Additionally, the documentation does not provide as much information as a user might require. For example, in the Ada/M(44) references, there is no Appendix F. (Appendix F lists the implementation-dependent characteristics.) As another example, the LRM states that the implementation defines whether or not a record component can overlap a storage boundary. The Ada/M(44) references do not specify whether or not storage boundary overlaps are allowed.

The following are samples of the specific compiler questions that are listed in [MeyersIIA 87] and their answers, which are taken from [MeyersIIIA 87] and [MeyersIVA 87].

1. What is the basic unit of SYSTEM.STORAGE_UNIT?

    a. VAX Ada - One byte or 8 bits.

    b. Ada/M(44) - One word or 16 bits.

2. What are the basic implementations of fixed point types?

    a. VAX Ada - Each fixed point type in VAX Ada occupies 32 bits. Values of fixed point types are represented as signed, twos complement (binary) numbers with an implicit binary scale factor.

    b. Ada/M(44) - Each fixed point type is stored as a right justified integer within 32 bits, with an implicit scale factor.

3. What are the basic implementations of floating point types?

    a. VAX Ada - Four implementations of floating point types are supported by VAX Ada. They are: FLOAT, represented as 32 bits which provide 6 (decimal) digits of precision; LONG_FLOAT, which may be represented as either 64 bits with 9 digits of precision or 64 bits with 15 digits of precision; and LONG_LONG_FLOAT, represented as 128 bits with 33 digits of precision. The choice of representation for the LONG_FLOAT type is determined by the use of a VAX-specific pragma.

    b. Ada/M(44) - One floating point type is offered: FLOAT with precision of 6 digits in the range -7.237005E75..7.237005E75. Type FLOAT is stored in two longwords (16 bits each). The least significant bit of the first word of the object must be bit 0 of an even memory address.

4. Does the compiler support the use of pragma PACK?

    a. VAX Ada - Yes.

    b. Ada/M(44) - Yes.

5. Does the compiler support the use of length clauses? What are the restrictions on their use?

    a. VAX Ada - Yes. There are no documented restrictions.

    b. Ada/M(44) - Yes. There are no documented restrictions.

4.2.3. Lessons Learned

Lessons learned from this effort include those listed below.

1. Know your compiler. The Ada language gives the compiler implementor many choices, especially in the area of representation clauses. The application programmer must know what implementation choices have been made for the compiler that is being used.
2. Be careful when referencing storage. When representing specific data placements, such as message formats, it is important to know the ordering schema for storage units and how bits are numbered within storage units. These are implementation dependent.
3. Be careful when using the SIZE attribute designator in a length clause. This also is implementation dependent.

The answers to the questions about the support provided by the two compilers indicate areas where one could expect implementation variation for other Ada compilers.

4.3. Low-Level I/O Experiments

A common characteristic of embedded software systems is a strong dependence on real-time input and output. Real-time input/output has unique properties in that it tends to be at the hardware device level, can be subject to strict timing requirements, and can be either synchronous or asynchronous in nature. Handling I/O for a specialized hardware device requires a special interface which has to provide all the capabilities typically found in device drivers and interrupt handlers. For example, a real-time application will need to enable, disable, and handle device interrupts; it may need to send control signals to and request status from a device; and it will most likely need to put and fetch data to and from a device's data register(s) or I/O memory.

This section provides a description of the experiments conducted in the area of low-level I/O and device drivers. In particular, we experimented with interfacing to three different devices, namely, a real-time clock, a serial I/O device, and a general-purpose, 16-bit parallel I/O device. The section discusses approaches taken in these experiments and summarizes the empirical results produced. Finally, the lessons learned from conducting these three experiments will be summarized.

4.3.1. Real-Time Programmable Clock Device Driver

The original intent of this experiment was to investigate various programming alternatives available to a real-time application developer for writing an interrupt handler and other Ada-callable routines for a programmable real-time clock. Furthermore, this experiment was to be be conducted in a way that would support a detailed analysis of the runtime costs associated with the various implementation alternatives. Specifically, measurements of the interrupt handler's execution speed, object code size, and interrupt latency were to be taken.

4.3.1.1. Approach

The approach was to code a simple Ada application in order to demonstrate that Ada can be used to implement interrupt handling functionally and to interface to a programmable real-time clock. The Ada software used for this experiment was to include the following:

- A main program that directs the real-time clock—either through an existing Ada-callable interface or through a newly developed one—to generate timer interrupts at a frequency of 500Hz. Within this framework, the main program was also responsible for opening and closing the log file, enabling and disabling the timer interrupts, establishing the connection between the clock's interrupt vector and the starting address of the service routine, and programming the clock rate.
- A simple application task scheduler that merely logs a message to an external text file each time it is called by the interrupt service routine.
- An interrupt service routine which handles the timer interrupts by invoking the application task scheduler.

Finally, upon successful implementation of this Ada software, the performance measurements of interest were to be made.

4.3.1.2. Accomplishments

To date, the real-time clock experiment has been completed for the VAXELN Ada 1.1 cross-compiler which targets a MicroVAX II processor running the VAXELN 2.3 kernel [VAXELN Release 86, VAXELN User's 85] and using a KWV11-C programmable real-time clock [LSI-11 86]. In practice, the experiment was limited to examining a single alternative for implementing an interrupt handler totally in Ada as no other Ada solutions were viable under this cross-compilation configuration. The solution involved using VAXELN kernel services to establish a link between the real-time clock's interrupt and the starting address of an interrupt service routine (ISR). Consequently, performing this experiment for the VAXELN target and cross-compiler involved implementing adequate Ada interfaces to the real-time clock [Borger 87a]. This was accomplished through two Ada packages. At the lowest level of the KWV11-C interface, the first package defines the necessary Ada data types (and their data representation specifications) to fully access and control the contents of the device's two 16-bit read/write registers. This package also implements two primitive operations for reading and writing the contents of the clock's control/status register. The second package provides the necessary data types, procedures, functions, and exceptions for interfacing to multiple KWV11-C real-time clocks via Ada application code. These routines support all four modes of the clock's operation as well as its five internal clock rates. In addition to providing a mechanism for establishing a link between clock interrupts and an interrupt service routine, this package supports typical programmable clock operations such as setting the clock's operation mode (e.g., repeated interrupts), setting the clock frequency, enabling and disabling clock interrupts, and programming the clock interrupt period. By using these two Ada packages, we developed the simple application task scheduler, including an ISR to handle the clock's interrupts. Sample Ada code was also developed to demonstrate how to make timing measurements using the real-time clock in its various operational modes, and the Ada interface was developed. The empirical results collected using these measurement techniques are summarized in the following section.

4.3.1.3. Empirical Results

We used two software measurement techniques involving the KWV11-C real-time clock to measure the elapsed time taken from when the interrupt is generated until the application code resumes execution after "waiting" for a device signal from an ISR.

Technique #1: The essence of this approach was to start at an interrupt frequency which the software could handle and to increase this frequency until the software could no longer service the interrupts fast enough. This gave a rough measure of the time elapsed from the interrupt occurrence until the application code was rescheduled and executed. This measurement was accomplished by operating the clock in Mode 1 and looping, decrementing the interrupt period by one for each iteration until the clock's overrun flag was set, which indicated that the software was not keeping up with the interrupt rate.

Technique #2: This technique was a more direct and reliable approach. It could be performed when the clock was operating in either Mode 2 or Mode 3. It combined the counter reading capability of Mode 2 or 3 with the fact that the counter will generate an interrupt when it overflows, regardless of the mode of operation. The approach was to enable counter overflow interrupts, start the counter, wait for a "signal" from the ISR caused by an interrupt, and finally read the current counter value.

VAXELN Ada Software Interrupt Latency (μsec)			
Measure	Technique 1	Technique 2/Mode2	Technique 2/Mode3
Maximum time	903.00	331.00	277.00
Minimum time	229.00	270.00	256.00
Average time	357.12	274.20	271.04
Standard Deviation	237.87	11.63	4.70

Table 4-1: VAXELN Ada Software Interrupt Latency Measurements

4.3.2. Serial I/O Device Driver

The purpose of this experiment was to demonstrate the use of Ada code in a "bare target" environment by developing a serial I/O device driver application.

4.3.2.1. Approach

The serial I/O experiment developed out of the configuration work required to install the SD Ada-Plus cross-compiler of Systems Designers, plc (SD). The experiment was conducted using the cross-compiler package version 2B.01. SD provided a serial device driver written in assembly code and configured for the target hardware. An attempt was made to rewrite the device driver in Ada.

4.3.2.2. Accomplishments

The serial driver was never completed because of the limitations placed on the length clause. The compiler would not allow a length clause for 8-bit integers. The hardware used for the serial port was a byte device, and the experiment was halted at this point.

4.3.3. Parallel I/O Device Driver

The purpose of this experiment was to demonstrate the use of Ada code in a VAXELN environment in a device driver application.

4.3.3.1. Approach

The first part of the experiment was an attempt to use VAXELN service calls with the DRV-11J general-purpose, 16-bit parallel interface board [DRV11 Reference Manual 79]. This attempt failed. The device did not respond as the documentation indicated it should. After some time was spent attempting to resolve this problem, the use of VAXELN service routines was dropped. The next step was to construct a device driver package, which required a detailed understanding of the DRV-11J device. The original intent of a general-purpose device driver was subsequently dropped. The device contained too many modes of operation to construct a package that would allow access to all modes of operation. Instead, the driver package has been used as a template and modified for each specific application.

4.3.3.2. Accomplishments

This experiment mutated from a pure experiment into an exercise in providing the application subgroup with a usable device driver for the parallel interface. A device driver package which meets their requirements has been produced and successfully tested. This package contains procedures for initializing, sending data, and receiving data. A collection of ISRs are also provided. No viable

programming alternatives were discovered. Timing measurements for performance analysis will be made after a hardware problem is corrected.

4.3.4. Lessons Learned

The following observations were made during the experimentation with VAXELN Ada and the real-time clock and parallel I/O interfaces.

VAXELN Ada 1.1

1. The VAXELN environment is not a bare machine environment. There is an operating system that tends to "get in the way." Application code is forced to use services provided by VAXELN instead of having direct interaction with the devices. The virtual addressing system is not a typical practice in embedded systems work. It is not completely documented, and translating from virtual to real addresses is an extra burden.
2. VAXELN suffers from its derivation from VMS. VAXELN was designed as an addition to VMS to provide more dependable time response to interrupts, but the system response time is very slow. A Digital representative informed us that VAXELN is really only suitable for "slow real-time systems." It is difficult, therefore, to see any application in the MCCR environment where VAXELN would be appropriate.
3. To redirect standard output to a file on a remote DECnet node, the File Access Listener option must be turned on at VAXELN system build time; furthermore, the file that will receive the output must exist with WORLD read and write access enabled. There are two alternatives for redirecting output: redefine the system logical SYS$OUTPUT at build time, or use Ada TEXT_IO routines (OPEN, PUT, PUT_LINE, CLOSE) with the remote file name.
4. When writing an interrupt service routine (or any Ada subprogram) that will be invoked by the VAXELN kernel, the following requirements must be satisfied to ensure proper runtime behavior [VAXELN Ada Release 86].

    - Each subprogram must either be a stand-alone program library unit or be declared at the outermost level of a library package (i.e., either in its specification or body).
    - The subprogram's name must be exported via the appropriate VAXELN Ada pragma (e.g., EXPORT_PROCEDURE) in order to resolve any external references during linking.
    - The subprogram must be compiled with a pragma SUPPRESS_ALL to disable stack overflow and underflow checks that would otherwise fail when invoked on the kernel stack.
    - The subprogram should avoid the use of Ada tasking operations and input/output operations, and should minimize the calls to external subprograms.

5. Using the /map and /full qualifiers on the EBUILD command yields a complete map of everything in a program's executable load module. This information is useful for examining the CSR and vector addresses of the known devices. It is also handy for learning which device drivers are being loaded along with the main program.
6. When building a VAXELN application that calls the CREATE_DEVICE service, the programmer must provide device-specific information in the program's VAXELN build file. The minimum that should be specified includes: the device name (a string which must match that used in the application's CREATE_DEVICE call), the CSR address, the interrupt vector address, and an indication as to whether or not to load the standard device driver. Additionally, the application must be built so that it can execute in kernel mode.

7. The VAXELN service KWV_INITIALIZE results in an access violation when used for re-initialization; the program terminates, which is incorrect behavior.
8. An Ada block with local variables whose memory locations are specified with address clauses provides an effective way of accessing data stored in particular locations of memory. For instance, the KWV_READ kernel service returns the starting address of the data it fetches. The following Ada code segment illustrates this technique for accessing this data, which is stored starting at a specific memory address:

```
KWV_READ (Identifier     => Clock_ID,
          Value_Count    => 1,
          Data_Array_Ptr => Clock_Data_Address,
          ST2_Go_Enable  => FALSE,
          Status         => Return_Code );

declare
  Ticks      : INTEGER := 0;
  Clock_Data : UNSIGNED_WORD;
    for Clock_Data use at Clock_Data_Address;
begin
  Ticks := INTEGER(Clock_Data);
  Put_Line(INTEGER'IMAGE(Ticks));
end;
```

9. There appear to be at least two alternatives for writing to and reading from device registers in memory: directly assigning locations and accessing them as Ada variables using the technique described above, or using predefined WRITE_REGISTER and READ_REGISTER subprograms. In practice, the first alternative cannot guarantee correct operational behavior—the generated code is likely to contain variable length bit field instructions which are not permitted by the architecture for accessing device registers. On the other hand, the WRITE_REGISTER and READ_REGISTER subprograms indicate to the compiler that only permissible instructions will be generated; and, therefore, the second alternative can guarantee proper runtime behavior.

SD Ada 2b.01

1. A number of observations about the SD system were made during the serial I/O device driver experiment. The product was not a good performer in terms of speed and size. The compilations were not brisk, and the SD libraries were quite large.
2. A bug was found and reduced to a simple loop for demonstration:

```
 1        procedure sdp1 is -- loop bug isolated
 2        subtype one_to_fifty is integer range 1..50;
 3        type array_type is array (one_to_fifty) of integer;
 4        x: one_to_fifty;
 5        array_1 : array_type;
 6
 7        begin
 8            x := 7;
 9            for index in x..x + 1 loop
10                array_1(index) := x ;
11            end loop;
12        end sdp1 ;
12
```

and the assembler code fragment =>

```
     0000005CP   0004
40   0000005EP   B081                    CMP.L     D1,D0
41   00000060P   6E50                    BGT.B     L3
42                           L5:
```

```
43  00000062P   356A                MOVE.W    2(A2),8(A2)
    00000064P   0002
    00000066P   0008
44                          L7:
45  00000068P   49EA                LEA.L     10(A2),A4
    0000006AP   000A
46  0000006CP   202A                MOVE.L    8(A2),D0
    0000006EP   0008
47  00000070P   0C80                CMPI.L    #50,D0
    00000072P   00000032
```

Lines 43 and 46 demonstrate the problem. Index is referenced as a word (16 bits) in line 43 and referenced as a longword (32 bits) in line 46. This problem was reported to SD customer support.

3. The pretty printer removed comments from source files. It did not default to "output comments" as stated in the documentation.
4. The TEXT_IO package provided the shell that allowed users to configure TEXT_IO to their hardware. Missing was a CROSS_IO package to allow terminal displays on the host computer's terminal. This feature was found in the other cross-compilers examined, and the vendor was informed that a CROSS_IO package is a useful item.
5. The cross-debugger was not able to perform a step command at either the Ada or machine code level. Program control was provided through the use of breakpoints at the Ada source level. This is a deficiency and should be corrected to make this product more useful.

4.4. Periodic Task Dispatching

The Ada tasking mechanism provides the real-time application programmer with a facility to do multitasking. The decision to use Ada multitasking depends mainly on the scheduling requirements of the application. Real-time applications can be classified by their inherent scheduling requirements into three categories [MacLaren 80]:

1. purely cyclic (periodic) scheduling with no aperiodic events
2. primarily cyclic with some aperiodic events and possible variations in computing loads
3. event-driven and no periodic scheduling

The Inertial Navigation System (INS) [INS Behavioral Specification 87, INS-tldd 87] simulator application being developed by the AEST Project must schedule both periodic tasks and aperiodic timeout tasks; its scheduling requirements therefore fall into the second category. Common practice has been to employ a cyclic executive for all three levels, but it has been shown that the benefits of Ada multitasking (e.g., supports aperiodic events, monitors intertask dependencies, controls task interaction, and supports cyclic processing at arbitrary frequencies) can be realized with applications having scheduling requirements falling into the latter two categories [MacLaren 80]. With Ada multitasking, the runtime system is responsible for scheduling tasks; whereas with a cyclic executive, the application programmer controls the scheduling.

Since the INS simulator falls into the second category above, we decided to use Ada tasking wherever possible in order to meet the application's scheduling requirements. The intent of this experimentation was to investigate various programming alternatives available to an application developer

for writing a real-time periodic task dispatcher in Ada. In an attempt to lessen the perceived risks of implementing the INS simulator using Ada tasks, the approach was to design and prototype alternative versions of a task dispatcher for the INS simulator so as to support a detailed schedulability analysis of the INS periodic task set. This section summarizes the results of that system modeling and analysis and offers some lessons learned from this prototyping effort.

4.4.1. Approach

In order to assess the schedulability of the INS periodic task set, the following four-step approach was taken.

Step 1 - Make real-time measurements

Prior to modeling the INS simulator tasking structure, it was essential to understand the internal operation of the underlying VAXELN [VAXELN Release 86, VAXELN User's 85] runtime executive. Key real-time measurements shown in Table 4-2 were either empirically obtained or taken from the VAXELN performance documentation.

Event	Time
Interrupt latency (VAXELN manual)	33 µsec
Context switch (VAXELN manual)	150 µsec
VAXELN signal/wait (empirical result, no process contention)	285 µsec
Ada rendezvous (empirical result)	1780 µsec
Attitude and Heading calculations (empirical result)	450 µsec

Table 4-2: VAXELN Real-Time Measurements

Step 2 - Estimate CPU utilization for task set

As a second step in the schedulability analysis, runtime estimates for each INS periodic task were made; execution time and CPU utilization estimates for the INS task set appear in Table 4-3. The execution time of the Attitude Updater was empirically measured to be 0.45 milliseconds, whereas the runtime for the remaining periodic tasks was estimated. The overhead associated with each periodic task represents the context switching time for entering and leaving the task (2 context switchs = 0.30 ms); for the Attitude Updater, the overhead represents the sum of interrupt latency and a context switch to the Dispatcher (0.03 + 0.15 = 0.18 ms). The synchronization times associated with each periodic task is 1.48 milliseconds, which is the measured Ada rendezvous time less 0.30 milliseconds for context switches. The 0.29 milliseconds of synchronization time for the Attitude Updater corresponds to the VAXELN signal/wait time (see Table 4-2).

Step 3 - Build INS tasking model

The third step of the analysis was the development of a skeletal INS tasking model. The control logic of each periodic task was virtually the same: an autonomous loop containing a synchronization point at the top followed by code to perform the task's computation. For the sake of modeling, the computational load of each periodic task was represented by a busy wait. For instance, the Velocity Updater task was instrumented with a 4 millisecond busy wait (see Table 4-3). To achieve the effect of

Task ID		Frequency (Hz)	Execution (ms)	Overhead (ms)	Synch (ms)	Execution Utilization (%)	Overhead Utilization (%)	Synch Utilization (%)	Total Utilization (%)
Attitude Updater		400	0.45	0.18	0.29	18.00	7.32	11.40	36.72
Velocity Updater		25	4	0.30	1.48	10.00	0.75	3.70	14.45
Attitude Sender		16	10	0.30	1.48	16.00	0.48	2.37	18.85
Navigation Sender		1	20	0.30	1.48	2.00	0.03	0.15	2.18
Status Display		1	100	0.30	1.48	10.00	0.03	0.15	10.18
Runtime BIT		1	5	0.30	1.48	0.50	0.03	0.15	0.68
Position Updater		0.8	25	0.30	1.48	2.00	0.02	0.12	2.14
Subtotals			164.45	1.98	9.17	58.50	8.66	18.03	85.19

Table 4-3: INS Periodic Task Set - Execution Time and CPU Utilization Estimates

varying the percentage of free CPU time, the duration of all of these busy waits could be scaled using a global load factor. For example, a global load factor of 0.75 is equivalent to the durations of each task's busy wait being 75% of its estimated value (0.75 * 4 ms = 3 ms for the Velocity Updater); a load factor of 1.25 increases the duration of the waits to 125% of their estimated values.

Step 4 - Monitor missed deadlines

The final step of the analysis was to vary the global load factor by increments of 0.05 and monitor the model behavior with respect to missed deadlines. For each dispatching technique under investigation, the global load factor was continually increased until a task deadline was missed. This critical load factor value, termed the schedulability threshold, was empirically determined for each dispatching alternative implemented.

4.4.2. Accomplishments

To date, the periodic task dispatching experiment has been completed for the VAXELN Ada 1.1 cross-compiler targeting a MicroVAX II processor that runs the VAXELN 2.3 kernel and uses a KWV11-C programmable real-time clock. For this particular cross-compilation configuration, a total of four different (prototype) periodic task dispatchers were developed. Two different periodic task dispatching approaches were used, referred to as the general-purpose queue manager (GPQM) and static queue manager (SQM); and for each of these, two different synchronization techniques were used, namely the Ada rendezvous (R) and the VAXELN semaphore (S) [Borger 87b]. Finally, empirical results of the schedulability analysis were collected, analyzed, and reconciled with theoretical expectations. The next section summarizes the key empirical results collected.

4.4.3. Empirical Results

The empirically collected schedulability thresholds for each prototype developed are summarized in Table 4-4.

		Base Calculations (%)	Context Switch (%)	Synch (%)	Dispatcher Execution (%)	Periodic Tasks (%)	Total Utilization (%)	Schedulability Threshold (%)
GPQM/R Estimate		18.00	14.66	18.03	12.80	40.50	104	**75**
GPQM/S Estimate		18.00	13.16	12.65	12.80	40.50	97	**100**
SQM/R Estimate		18.00	14.66	18.03	8.80	40.50	100	**85**
SQM/S Estimate		18.00	13.16	12.65	8.80	40.50	93	**110**

Table 4-4: Estimated CPU Utilizations and Schedulability Thresholds

Interpretation of the schedulability threshold data in Table 4-4 indicates that, assuming the same synchronization mechanism, changing from the GPQM Dispatcher to the SQM Dispatcher yields a 10% increase in the schedulability threshold. Additionally, interpretation of the schedulability threshold data in Table 4-4 indicates that, assuming the same dispatching approach is used, a 25% increase in the schedulability threshold results if the synchronization mechanism is changed from the Ada rendezvous to a VAXELN semaphore. Finally, this data shows that a 35% improvement in the schedulability threshold occurs when changing from the GPQM Dispatcher and the Ada rendezvous for synchronization to the SQM and VAXELN semaphores.

It can be computed from the data in Table 4-4 that there is a 7% difference in total CPU utilization when varying the synchronization mechanism used by the Dispatcher. This implies that using VAXELN semaphores for task synchronization uses 7% less CPU time than Ada rendezvous for this real-time periodic task dispatcher application.

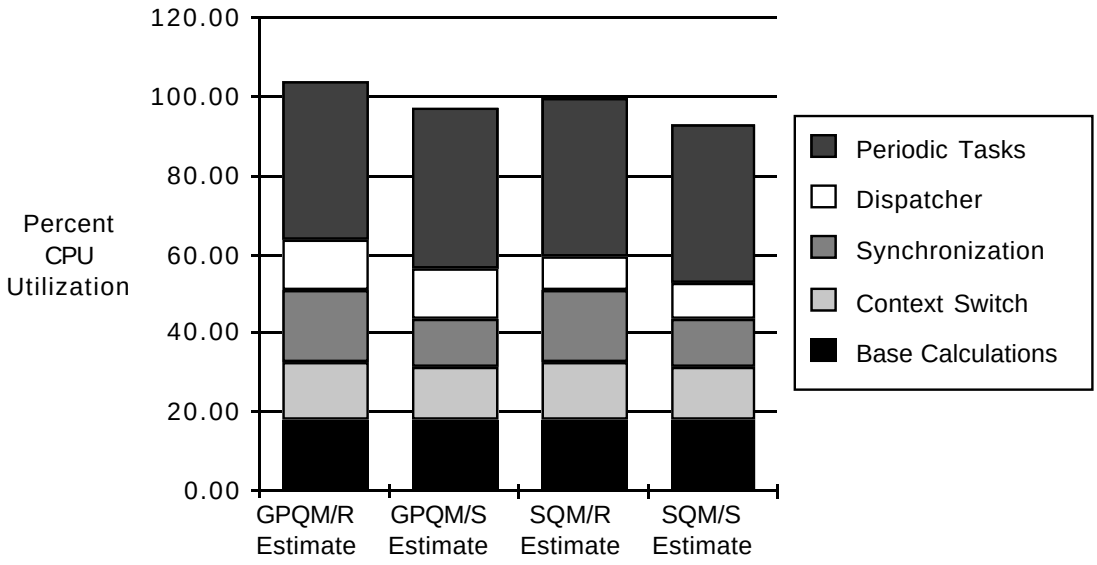

Figure 4-1: Rendezvous Versus Semaphore Comparison

Since the (estimated) execution times of the INS simulator's base calculations and periodic tasks are constant, Table 4-4 can be used to illustrate the implications of using the synchronization mechanism for scheduling the periodic tasks. The bar chart (generated from this data) in Figure 4-1 clearly illustrates the pervasive effect of the Ada rendezvous on the context switch, synchronization, and dispatching CPU utilization.

4.4.4. Lessons Learned

The total estimated CPU utilization was quite high for the interrupt service routine (Attitude Updater) and the periodic task, without including the empirical results for the Dispatcher's utilization. When Ada rendezvous was used for synchronization, it was 85% (see Table 4-3); and for VAXELN semaphores, it totaled 78%. It is clear from the empirical evidence that a savings of 11% CPU utilization would have been gained if the synchronization between the ISR and the Dispatcher were

eliminated. This could be simply done by moving the Dispatcher's responsibilities into the ISR. In practice, however, that was not possible since numerous VAXELN Ada ISR restrictions (see Section 4.3.4) limited the number of Dispatcher implementation alternatives.

The empirical results illustrate the pervasive effect of the Ada rendezvous on the schedulability of the INS task set. We found that using the Ada rendezvous for synchronizing between the Dispatcher and the periodic tasks rather than VAXELN semaphores results in a 7% increase in total CPU utilization, regardless of the dispatching technique employed. Furthermore, for both dispatching methods implemented, given the original execution time estimates for the INS periodic tasks, using the Ada rendezvous as the synchronization mechanism results in missed task deadlines. Only when these estimates are scaled by 75% and 85% for the GPQM and SQM dispatching approaches, respectively, does the task set become schedulable with Ada rendezvous for task synchronization. Interpretation of the schedulability threshold data further demonstrates the impact of the Ada rendezvous on the task set schedulability. The empirical results show that, assuming the same dispatching approach is used, a 25% increase in the schedulability threshold results if the synchronization mechanism is changed from the Ada rendezvous to a VAXELN semaphore; furthermore, a 35% improvement is obtained when changing from the GPQM Dispatcher and the Ada rendezvous for synchronization to the SQM and VAXELN semaphores.

The schedulability thresholds determined empirically were consistent with those computed theoretically. For example, given the original execution time estimates for the INS periodic tasks, the SQM dispatching approach using VAXELN semaphores for task synchronization yielded a total CPU utilization level of 93%. Furthermore, it was found empirically that the task set was schedulable until the original time estimates of the periodic tasks were scaled by 1.1 or until the total CPU utilization level reached 97% (ISR + Scaled Periodic Tasks + Dispatcher = 37 + 1.1 * 41 + 15 = 97.1%). Similarly, solving for the schedulability threshold using the task-lumping method [Sha 87] results in an expected threshold value of 1.12.

5. Application Development

5.1. Background

5.1.1. Purpose

The development of an application is the third component in the project's study of Ada in the real-time embedded arena. It serves to provide the project with credibility in the embedded systems world. It also provides a context for using information and results generated by the project, and the completed application is an Ada artifact that may continue to be used. Perhaps the most important purpose of the application is to serve as "a proof of concept that Ada can be used for the design and implementation of time-critical MCCR applications" [Weiderman 87b].

To ensure the credibility and relevance of project results, it is important that our context for studying Ada closely approximates the context in which Ada will be used. Building a subset of an actual real-time embedded application in Ada helps to ensure a proper context. With the help of a resident Navy affiliate from the Naval Surface Weapons Center (NSWC) in Dahlgren, VA, we have been able to create the specification for an Inertial Navigation System (INS) simulator that preserves the real-time properties of an actual onboard INS while greatly simplying those aspects of the system that require a high degree of application domain knowledge.

In order to check the relevance of data obtained from benchmarking and real-time experiments, the application group serves as a potential recipient of the results. Indeed, this interaction has proved to be quite valuable. In addition, the application effort will undoubtedly generate additional issues and questions that were not considered during the initial phase of the project. Once the application is completed, it can be used as a vehicle to look at transportability issues, and it can serve as a composite benchmark to assess performance and overall behavior of Ada runtime systems.

The design and implementation of this application is an experimental effort. The main goal is to record lessons concerning the use of Ada in the real-time embedded arena. We are not bound by any particular design methodologies or software development processes.

5.1.2. Beneficiaries

The Ada community at large will benefit from the experience gained as a result of this development. The MCCR community, which is concerned with construction of real-time embedded systems in Ada, is targeted to benefit most from this work. In general, the construction of an embedded application should uncover mature and immature technologies and the risks associated with using language features and programming paradigms. The MCCR community is also interested in the general problem of using Ada with real-time scheduling and distributed processing, which are intended areas of investigation. In addition, Ada compiler vendors should be able to benefit by having information that allows them to better serve the needs of embedded system builders.

5.1.3. Progress to Date

Since an incremental development approach has been taken, design and development has proceeded further in some areas than in others. The INS simulator is comprised of seven subsystems: Main, User Interface, Executive, Communications, Data Extraction, Motion Simulation, and Built-In-Tests [INS-tldd 87]. Several of the INS subsystems have reached the detailed design level; and in several areas code has been written while prototyping, including the executive and device drivers for a real-time clock and for a serial interface. Code has also been written for portions of the user interface and communications subsystems. The other subsystems are in the initial phase of detailed design. Effort has not yet been expended on the external computer, with the exception of informal cursory analysis assessing the reusability of portions of the INS simulator. A gross estimate indicates that about half of the INS simulator code should be reusable.

5.2. Specifying an Application

5.2.1. Criteria for Application Selection

The criteria for selecting a suitable application are enumerated in [Weiderman 87b]. The essential criteria are:

- Functional and performance requirements must be representative of embedded systems in the MCCR community. The application should be characterized by strict timing requirements and must interact with multiple devices, thus requiring that scheduling concurrent activities and writing device drivers be part of the problem.
- The application must be a large enough subset of a real application to be credible but small enough to be completed in one to two man-years.

5.2.2. Tailoring an Application

It would have been extremely difficult and time consuming to find a suitable application that meets all of the specified criteria. The pragmatic solution was to capitalize on the availability of our Navy affiliate, an expert in the inertial navigation domain, and tailor an existing application to meet our needs. The strategy behind tailoring is to remove unnecessary domain specificity while preserving the real-time essence of the system.

As seen in Figure 5-1, the INS computer is one in a series of computers whose role is to gather and filter sensor data which represents ship motion in terms of attitude, position, and velocity. This information is sent through a series of processors to a launch-sequencing processor that controls the firing of onboard missiles.

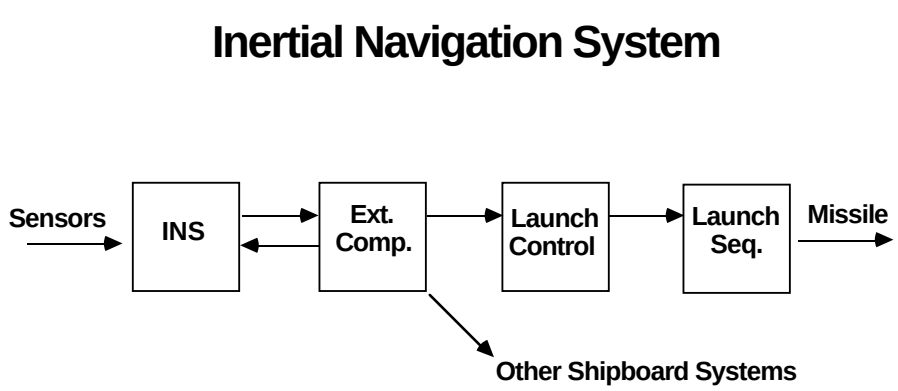

Figure 5-1: Inertial Navigation Data Flow

Our goal is to design and implement an INS simulator. The INS simulator compares to the INS as follows:

- The INS uses sensors and gyros to determine ship motion; the INS simulator uses uncoupled sinusoids as a model to internally generate ship motion.
- The INS uses Kalman filtering to process raw sensor data; the INS simulator generates its own motion data, which is not filtered.
- Both implement the message-passing protocol to communicate with the external computer.
- Both have periodic processing requirements with periods that range from 2.5 milliseconds to 1000 milliseconds and time-out requirements of 5 and 10 milliseconds.

The role of our resident affiliate became particularly important to the task of minimizing the domain-specific characteristics and unnecessary details of the application. This tailoring simplified the application without sacrificing its real-time properties and also eliminated the need to rely on a domain expert. We could then concentrate on design and development with a specific focus on characteristics Ada environments and runtime systems.

5.3. Inertial Navigation System Application Design and Development

5.3.1. Development Philosophy

The development of the INS application has been based on the following three guidelines:

- maximize use of Ada
- experiment and prototype
- design for portability

5.3.1.1. Maximize Use of Ada

The INS application is not an end in itself, but a vehicle for investigating the practical issues involved in designing and developing a real-time embedded system in Ada, using currently available Ada runtime systems. Thus, the first guideline calls for using Ada constructs wherever appropriate. This applies particularly to those Ada features, such as tasking, which are designed to facilitate the development of embedded system software.

Of course, if a particular (optional) Ada feature is not implemented, or if it is implemented so inefficiently that the real-time constraints of the application cannot be met, then it will be necessary to resort to explicit calls to the runtime system services (if they exist) or to assembly language. One of the goals of this development effort is to uncover and document these potential difficulties.

5.3.1.2. Experiment and Prototype

The second guideline calls for performing some experimentation and prototyping prior to, or in conjunction with, the final design for each implementation. The aim is to reduce the risk of producing a design that cannot be implemented or that will not satisfy the requirements. This is particularly important since the design team had no direct experience in using the proposed cross-compilers to develop real-time Ada programs for the proposed target microprocessors. The particular areas of concern are precisely those areas that require Ada features that are implementation-dependent, such as

- building device drivers
- scheduling tasks
- writing representation specifications

5.3.1.3. Design for Portability

Once completed, the INS simulator is to serve as a composite benchmark of Ada runtime systems. It will be ported to a variety of target processors, probably using a variety of cross-compilers and runtime systems. This means that certain details of the program (e.g., interfaces to the hardware and the runtime system) will be different for each implementation. Thus, the third guideline calls for producing a general design with these foreseeable differences encapsulated in distinct, implementation-dependent modules which are accessed via consistent virtual interfaces. We also expect to derive a portion of the external computer system from the INS simulator program.

5.3.2. Development Strategy

Based on the above development philosophy, and apart from the technical details of Ada design, certain overall methods were employed in the process of producing the INS design and partial implementation. These are explained in the following paragraphs.

5.3.2.1. Pre-Design Experimentation

Even before the INS application was selected, we wrote some small experimental programs on the VAXELN compiler system to familiarize ourselves with this style of host/target development system. The experimental programs were mostly benchmarks of certain Ada features that seemed to be relevant to a typical embedded system application (e.g. clock resolution and tasking overhead). This early experimentation exposed the need for a real-time clock and, in general, supported the value of early experimentation and prototyping.

5.3.2.2. Prototyping

In the early stages of the design, there was much reliance upon the other AEST subgroups for experimental information about the availability and performance of important implementation-dependent Ada features on the various target systems.

During the later stages of the design, there was further cooperation in the prototyping of critical areas:

- periodic task dispatcher
- device drivers
  - programmable real-time clock
  - 16-bit parallel interface
  - user terminal interface
- communications link protocol
- operator interface

5.3.2.3. Interaction with Carnegie Mellon in the Area of Scheduling Theory

The deliberate choice of a "hard real-time" application meant that the proper scheduling of the various internal tasks was an important and difficult issue. Fortunately, the Advanced Real-Time Systems (ART) Project, a small research group at Carnegie Mellon, is working in this field; and a useful dialogue was established. In particular, our initial common-sense hierarchy of Ada task priorities was shown to be equivalent to a known theoretically optimal solution discussed in [Liu 73]. There have also been discussions of various practical refinements and of certain theoretical problems with the Ada tasking paradigm (e.g. priority inversion) [Cornhill 87]. There is ample scope for continued collaboration.

5.3.2.4. Reviews

In order to check the quality of the design and the adequacy of the documentation, a number of design reviews were held. Several SEI staff members from outside the project participated in informal reviews (preliminary design review on 25 March 1987 and detailed design review on 5 June 1987). A slightly more formal review was held at the Naval Weapons System Command in Dahlgren, VA, (the home laboratory of our Navy affiliate) on 21-22 July 1987.

5.3.3. Documentation

The design and development of the INS application has generated the hierarchy of documents listed below.

5.3.3.1. Requirements Specifications

The INS application is based on the following requirements specifications, written by the resident Navy affiliate and subject to our review:

1. System Specification [Meyers 87]
2. Inertial Navigation System Simulator Functional/Performance Specification [Meyers-INS 87]
3. External Computer System Functional Specification [Meyers-EC 87]

5.3.3.2. INS Behavioral Specification

As a result of feedback from reviewers, an additional document, the Inertial Navigation System Behavioral Specification [INS Behavioral Specification 87], was written to supplement and clarify the original specifications. A comparable document for the external computer system is planned.

5.3.3.3. Top-Level Design Documents

The general top-level design of the INS simulator program is described in [INS-tldd 87]. A comparable document for the external computer system is planned.

5.3.3.4. Detailed Design Documents

The detailed design document for the INS simulator program is not yet complete, but there are design overviews of three critical areas (clock dispatcher, communications link, and user interface).

5.3.4. Design Methods and Issues

No single, specific design methodology was used to design the INS simulator. Rather, the project applied a general cognizance of relevant design methods, including MASCOT as discussed in [Allworth 81], DARTS [Gomaa 84], and object-oriented design [Booch 86]. Our overall approach is probably best characterized by Nielson and Shumate [Nielson 87]. Our development methodology is complemented by rapid prototyping and incremental development. This was consistent with our underlying goal of creating an Ada artifact to use in later experimentation. Incremental development allows us to create intermediate Ada artifacts that are usable for experimentation, and it underscores the relative importance placed on the the product that we are producing as opposed to strict adherence to a specific development process.

We began by choosing a set of data transforms to model the fundamental objects in the domain. The set of data transforms was then mapped onto a set of tasks, and intertask control structure was determined. The Ada packaging structure is being incrementally constructed.

5.3.4.1. Data Flow

The process of determining data flow involved describing the major data transforms, data stores, and the data flows between stores and transforms. Data transforms are active objects such as the Message Encoder or the Command Window Processor; they correspond to real world entities. Data stores are passive objects, essentially representing tables of data or queues (essentially abstract data types). Data flows show the data interdependencies.

A remaining area for experimentation is implementation of guarded access to common data stores, which may be implemented via an Ada monitor task or a hardware-critical section. The performance tradeoffs between these two alternatives must be examined.

5.3.4.2. Identification of Tasks

Given a set of data transforms, the identification of tasks was guided by the following criteria:

- Is the role of the data transform that of a server where the clients may proceed independently after synchronization?
- Can the data transform be viewed as an actor [Borger 86] with periodic processing requirements?
- Is the data transform an interrupt service routine?

- Is the primary role of the data transform that of a producer or consumer of I/O to or from a device?
- Are the functions of the data transform time critical?
- Can the functions of multiple data transforms be grouped together?

The mapping between data transforms and tasks was not simple. In some cases, it was one-to-one and other cases many-to-one and one-to-many.

When designing the concurrency architecture of the INS simulator, we followed a philosophy that dictated maximizing the use of Ada features even when we knew that there would be a performance penalty. This resulted in the real-time architecture exhibited in Figure 5-2. Notice that tasks have been grouped into three main processing levels. The broad levels of processing used throughout this section are those of [Allworth 81]:

- Interrupt Level: highest priority tasks that serve as interrupt service routines.
- Clock Level: tasks with a cyclic control structure that are activated via an entry call from the Dispatcher.
- Base Level: the remaining tasks, which are executed in the background and include servers to clock-level tasks and producers and consumers of I/O that interact with interrupt level tasks.

This philosophy has opened the door for performance enhancements in many areas. Part of our exploratory effort has been to document the areas where we were forced to migrate to more efficient solutions, several of which are currently being explored. They are documented in the top-level design document [INS-tldd 87].

5.3.4.3. Real-Time Dispatcher

After the concurrency architecture was designed, a basic problem of scheduling still remained: the problem of meeting periodic requirements as small as 2.5 milliseconds in the face of current Ada runtime systems, whose resolution of time is 10 milliseconds, which does not guarantee schedulability of a task at the expiration of the Ada delay. In order to meet the periodic scheduling requirements, a real-time clock and task dispatcher had to be introduced. Referring to Figure 5-2, the real-time dispatcher consists of a hardware real-time clock (that can generate interrupts at frequencies up to 1 megahertz), a Task Dispatcher and an Activation Queue Manager. The details of the real-time manager are discussed in Section 4.4.

Basically, the real-time clock is programmed to interrupt every 2.56 milliseconds (the periodic requirement with the smallest period). Upon the handling of the interrupt, the Task Dispatcher receives control. Using services of the Activation Queue Manager (AQM), the Task Dispatcher determines if there is a task that must be activated at the current tick and, if so, activates it by synchronizing with the periodic task. In addition, the time-outs required by the INS communications protocol are facilitated through the use of a Time-Out Server that also works in conjunction with the AQM and the Task Dispatcher.

Task priorities are assigned using the rate-monotonic discipline [Liu 73], assigning priorities according to task execution frequency, where tasks with highest execution frequency receive the highest priority. For a strictly periodic task set, this guarantees schedulability when processor utilization is below 69 percent.

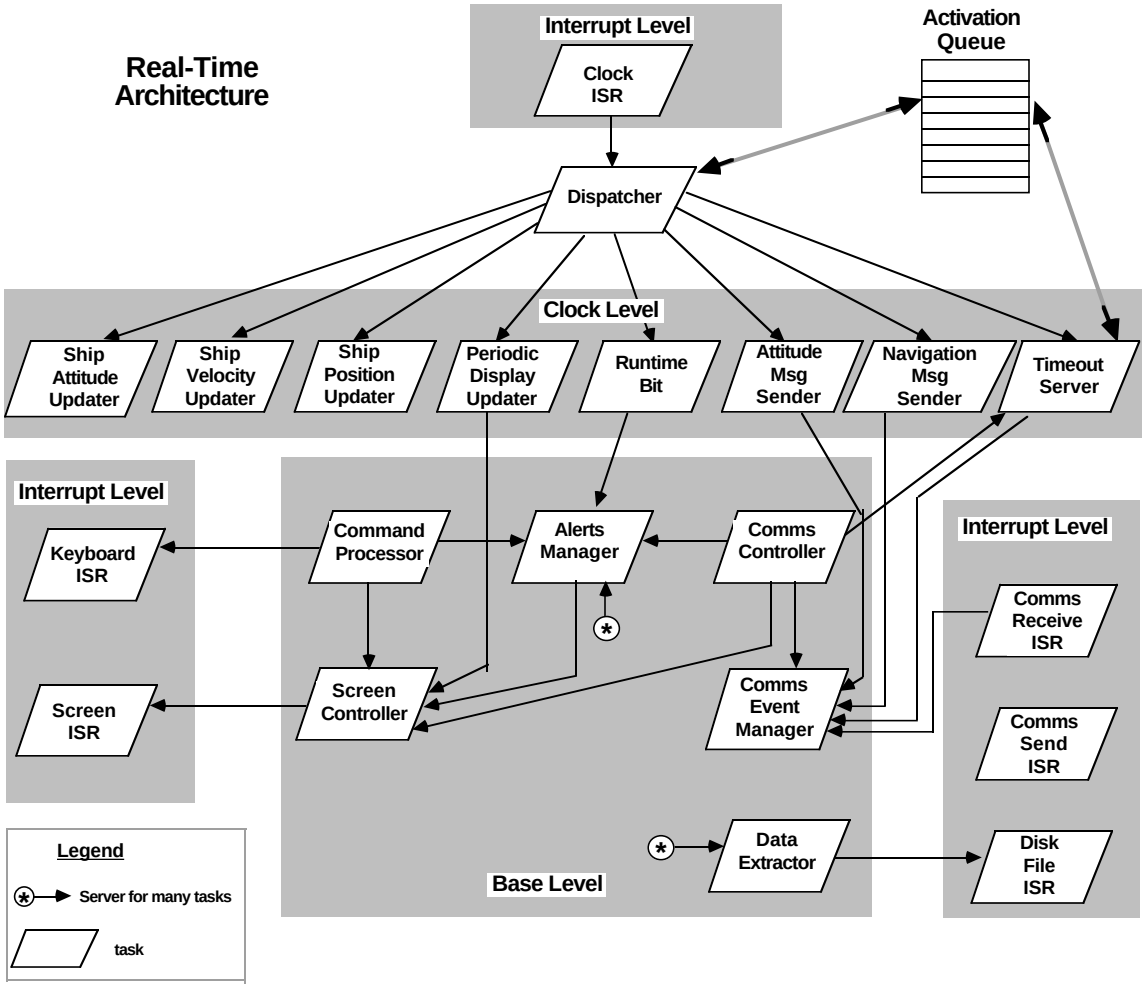

Figure 5-2: Real-Time Architecture

It is important to note that the real-time dispatcher works in conjunction with the Ada runtime system scheduler, not in place of it. The real-time dispatcher merely synchronizes with a task that has suspended itself in anticipation of subsequent activation.

5.3.4.4. Encapsulation of Communications Protocol

The communications protocol is implemented as a set of nested state tables, each state table being encapsulated in a subprogram. Many of the state transitions are triggered by events that must be communicated to this task from other tasks; for example, the arrival of a communications control word from the external computer or a notification of a time-out from the Time-Out Server. The vehicle for this is an entry call. Since Ada accept statements cannot be embedded in subprograms, another task had to be introduced to handle events.

5.3.4.5. Efficiency Versus Portability

As previously described, the initial design of the INS simulator is one that attempts to take full advantage of Ada tasking and Ada features in general. This approach was motivated by portability concerns and the general premise that the charter of the project is to learn about Ada. The next question we must ask is: If we cannot meet performance requirements using "pure Ada" (either because features are implemented inefficiently or because they are not implemented), how far do we have to retreat to meet the performance requirements that have been imposed by the application? Section 4.4 reports on preliminary findings in the area of task synchronization. These findings indicate that using Ada tasking in conjunction with faster task synchronization mechanisms (provided by an auxiliary executive or runtime system service) may represent a viable compromise between using Ada tasking and efficiency. This finding is consistent with suggestions offered in the *Ada Adoption Handbook* [Foreman 87].

5.4. Lessons Learned Designing the INS Application

5.4.1. Finding an Application

One would think that there are many candidate real-time embedded applications that satisfy the criteria outlined in Section 5.2.1. In fact we found it extremely difficult to find an existing application that lent itself to our needs. To reduce the risk inherent in our search, we attempted to specify our own application. That, too, proved to be difficult and met with criticism that it was not a credible application. The solution—and a key lesson—is that a domain expert is needed to help interpret or respecify the requirements for a credible application and to facilitate the removal of domain specificity. In this case, we were able to use the expertise of our Navy affiliate, who extracted the essential real-time requirements from an existing application and wrote several requirements specification documents for the INS simulator and the external computer.

5.4.2. The Ada Language

There are several language features that had an impact upon the design.

This is a well-known problem: the semantics of the Ada delay statement require a delay that is at least as long as one specified (not precluding an actual delay, which is considerably longer). This type of behavioral ambiguity could not be tolerated when designing the INS simulator, which needed a real-time clock and the task dispatcher regime. A task executing a delay statement should be rescheduled within SYSTEM.TICK seconds of the expiration of the specified delay. However, for most commercially available runtime systems, even this delay resolution is not fine enough for the INS simulator; but it would be a step in the right direction.

We call a set of related packages a subsystem and allow some of these packages to be visible to other subsystems. This is purely convention, with no language support and no implementation support. In several cases, however, we attempted to mimic subsystems by nesting packages inside of packages and found this to have undesirable side effects. One side affect is that relationships between packages nested in a global package become implicit instead of explicit. The visibility relationships between multiple packages which are nested in a higher level package are determined strictly by their ordering. The second side effect was an unnatural layering of packages. This approach was

abandoned in favor of creating separate packages at the higher level and adopting the subsystem convention mentioned above.

5.4.3. Ada Implementations

Most of the lessons learned concerning specific Ada implementations have been discussed in Sections 4 and 5. However, two specific lessons have particular relevance to the application and will be highlighted here. Both concern Chapter 13 machine-dependent features.

Under VAXELN, fixed and floating point types use 32 bits regardless of the specified length clause. This affects the construction of messages in the INS, where a scaled integer must be placed into 20 bits.

We found severe limitations concerning the allowable functionality from within an ISR under VAXELN. Specifically, you are prohibited from making calls to any of the runtime libraries. This may prohibit the placement of the attitude calculations into the ISR, which may have a significant performance impact.

The general lesson is that the machine-dependent features have amplified importance in embedded systems, yet this is the area where compilers tend to be the weakest or have the most restrictions.

5.4.4. The Design Process

In Section 5.3.4 we enumerated several of the design methods from which we borrowed ideas and concepts. In addition, we used the statechart [Harel 87] representation for state transition diagrams to more formally specify the communications protocol. The application of state transition diagrams to the communications protocol was natural; however, this method did not lend itself to other segments of the application. The general lesson is that one should be judicious in applying methods and representations when designing each subsystem.

Ada is being informally used as a program design language (PDL). Subjectively speaking (we have not done an analysis of PDLs) we have found it helpful to use the implementation language as a PDL. However, it is not sufficient in its ability to convey a design and does not take the place of data flow diagrams and control and concurrency diagrams. Also, we found a tendency to make the design too detailed too soon (that is, to code before we really were ready or before we had intended to). In addition, one can get a false sense of security concerning the design simply because the PDL has successfully compiled.

The value of prototyping cannot be overstated, especially in an experimental setting such as this. It has been necessary to explore many areas in order to build the INS application. These include: peculiarities of the Ada implementation; unknowns regarding scheduling and time budgets; and the need to write device drivers. Since they all have an impact upon the design, it is extremely valuable to have experimental prototyping concomitant with the ongoing design effort. It greatly reduces risk by exploring the validity of design decisions, offers early performance feedback, and develops an early cognizance of the nuances of the machine dependencies of the Ada implementation.

It should also be noted that the concurrency architecture is not evident from the Ada packaging strategy. For example, a package may be used to implement an abstract data type such as a circular

buffer. The package will export *put* and *get* subprograms. These subprograms, although part of the same package, may be used in different threads of control. The lesson is that the control and concurrency view of the system should not be confused with the packaging architecture view.

5.4.5. The Review Process

The design and development of the application was informal in the sense that we were not saddled with any particular development methodology or process, but it was formal in the sense that we adhered to good software engineering practices, periodic reviews included. We conducted several design reviews at the SEI and one review at NSWC in Dahlgren, VA. In addition to reaping the benefits of technical peer review, we learned several lessons about conducting reviews.

Our main problem arose from the incipiency of the organization in this particular area; we had no corporate policy or procedure for conducting design reviews. In addition, software engineering is a relatively immature discipline and lacks generally accepted review practices. Since the SEI is comprised of a heterogeneous population, different reviewers—and the reviewees—had different notions of the review process. It therefore became extremely important to be very explicit in communicating goals and expectations for each review. To compound the problem, the group of reviewers had to become at least slightly familiar with the application domain. This made continuity of reviewers extremely important.

One of the important products of the SEI reviews was a reorganization of our documentation strategy. To help isolate the reviewers and ourselves from domain specifics and to solidify ambiguities in the functional and performance specification documents, a review team recommended that we write a behavioral specification. Another documentation-related criticism was that the top-level design document varied in its level of description from subsystem to subsystem. As a result, we wrote a new top-level design document. The review at Dahlgren was also helpful. It was important to confirm that our approach was reasonable to professionals in the application domain. It was also important to verify the relevance of attempting a proof of concept for Ada in a real-time embedded domain.

5.5. Future Work

Future work includes completing the INS simulator, reusing as much code as possible from the INS simulator to complete the external computer, and porting the INS simulator first to the Motorola 68020 processor and then to the MIL-STD-1750A Instruction Set Architecture processor.

6. Summary and Future Directions

This report has summarized the efforts of the AEST project. The emphasis has been on four major components of this effort:

- Establishment of a testbed environment, consisting of hardware, software, test tools, and Ada compilers to support the project.
- Benchmarking to quantitatively assess performance of Ada runtimes.
- Experimentation to examine critical factors affecting the use of Ada in typical real-time embedded applications.
- Development of a prototypical embedded application with hard real-time requirements to demonstrate the use of Ada in these environments.

Following the investigative approach initiated in 1986-87, the work in the 1987-88 time frame will concentrate on three aspects of the project:

- expanding the testbed to new targets
- testing Ada in distributed homogeneous multiprocessor configurations
- transitioning the results and lessons learned to others

In the area of hardware, two new target processors are planned. The first will be the MIL-STD-1750A, which is being used as the 16-bit processor for all Air Force avionics systems as well as some Army avionics systems. The second processor will be the Intel 80386. This processor is the fastest and most recent in the Intel iAPX86 family and is being considered for a number of weapon systems. For host computing, the AEST Project will incorporate the Rational 1000 and its Ada environment. The Rational is already part of the SEI computing environment, and we will have cross-development systems for the 1750A in the first quarter of 1988 and for the MC68020 in the second quarter of 1988. The third cross-development system will be for the VAX and is scheduled for later in 1988.

Software additions planned include three cross-compilers for the 1750A (including the Rational cross-compiler), one for the Intel 80386, and one additional cross-compiler (probably Rational) for the MC68020. In addition, a real-time executive will probably be required for testing Ada in distributed environments. One possibility is Ready Systems MPV (Multiprocessor VRTX [Versatile Real-Time Executive]) for the Motorola 68000 family. The emphasis will be placed on using off-the-shelf products for these investigations rather than trying to develop our own software.

In the area of benchmarking, we will run selected groups of tests from several benchmark test suites to explore the time, space, and capacity constraints associated with individual Ada features. The PIWG and University of Michigan suites are already in-house. The ACEC test suite and the Ministry of Defense (United Kingdom) test suite will be added during the year. Gaps in the coverage of these existing test suites will be identified, and new tests will be developed. Of particular interest is the area of distributed processing and hard real-time scheduling. There are few prototypical problems that address these areas. These investigations will be undertaken with the Advanced Real-Time (ART) Project at the Carnegie Mellon Computer Science Department.

We will continue to design, implement, and run a series of experiments to explore programming alternatives, programming idioms, implementation approaches, and real-time ramifications of using Ada. These experiments will be continue to be driven by the needs of MCCR systems in general and our INS application in particular. Primary areas of focus will be: machine-dependent features of Ada, real-time scheduling, and low-level I/O.

Finally, we will complete the implementation of the INS application as an example of Ada used for real-time embedded systems. The application system will provide the context for using experiment and benchmark information to investigate programming alternatives and will be the primary vehicle for investigating the issues regarding the porting of an Ada application to other target processors and to distributed target processors. At present, it is still questionable whether the application will meet its timing deadlines on the VAX configuration. The first port of the system will be to the MC68020 target. We will continue to tune the application for performance and to simultaneously look for faster and more highly optimizing compilers.

The AEST Project has learned much in its first year, but there are still many concerns about the language, the implementations, the tools, and the use of the language in MCCR systems. We will continue to try to shed more light on the issues and try to accelerate the process of making Ada the language of choice for developing real-time embedded applications.

References

[Allworth 81] Allworth, S.T.
Introduction to Real-Time Software Design.
MacMillan, London, 1981.

[Altman 87a] Altman, N. A. and Weiderman, N. H.
Timing Variation in Dual Loop Benchmarks.
Technical Report CMU/SEI-87-TR-21, Software Engineering Institute, October, 1987.

[Altman 87b] Altman, N. A.
Factors Causing Unexpected Variations in Ada Benchmarks.
Technical Report CMU/SEI-87-TR-22, Software Engineering Institute, October, 1987.

[Booch 86] Booch, G.
Software Engineering with Ada.
Benjamin Cummings, 1986.

[Borger 86] Borger, M. W.
Ada Task Sets: Building Blocks for Concurrent Software Systems.
In *Proceedings of the IEEE Computer Society Second International Conference on Ada Applications and Environments*. Miami Beach, FL, April, 1986.

[Borger 87a] Borger, Mark W.
VAXELN Experimentation: Programming a Real-time Clock and Interrupt Handling Using VAXELN Ada 1.1.
Technical Report CMU/SEI-87-TR-29, Software Engineering Institute, October, 1987.

[Borger 87b] Borger, Mark W.
VAXELN Experimentation: Programming a Real-Time Periodic Task Dispatcher Using VAXELN Ada 1.1.
Technical Report CMU/SEI-87-TR-32, Software Engineering Institute, December, 1987.

[Broido 87] Broido, Michael D.
Response to Clapp et al: Toward Real-Time Performance Benchmarks for Ada.
Communications of the ACM 30(2):169-171, February, 1987.

[Clapp 86] Clapp, Russell M., et al.
Toward Real-Time Performance Benchmarks for Ada.
Communications of the ACM 29(8):760-778, August, 1986.

[Cornhill 87] Cornhill, D.T., Sha, L., Lehoczky, J.P., Rajkumar, R., Tokuda, H.
Limitations of Ada for Real-Time Scheduling.
In *Proceedings of the International Workshop on Real-Time Ada Issues, Moreton-hampstead, Devon, UK, 13-15 May 1987*. Special Edition of Ada Letters, Fall, 1987.

[Curnow 76] Curnow, H. J. and Wichmann, B. A.
A Synthetic Benchmark.
The Computer Journal 19(1):43-49, February, 1976.

[Digital 85] *VAXELN User's Guide.*
Digital Equipment Corporation, 1985.

[Digital 86a] *VAXELN Ada User's Manual.*
Digital Equipment Corporation, 1986.

[Digital 86b] *VAXELN Release Notes.*
Digital Equipment Corporation, 1986.

[Donohoe 87a] Donohoe, P.
A Survey of Real-Time Performance Benchmarks for the Ada Programming Language.
Technical Report SEI-87-TR-28, Software Engineering Institute, December, 1987.

[Donohoe 87b] Donohoe, P.
Ada Performance Benchmarks on the MicroVAX II: Summary and Results.
Technical Report SEI-87-TR-27, Software Engineering Institute, December, 1987.

[Donohoe 87c] Donohoe, P.
Ada Performance Benchmarks on the Motorola MC68020: Summary and Results.
Technical Report SEI-87-TR-40, Software Engineering Institute, December, 1987.

[DRV11 Reference Manual 79]
DRV-11J Parallel Line Interface User's Guide.
Digital Equipment Corporation, 1979.

[Foreman 87] Foreman, J. and Goodenough, J.
Ada Adoption Handbook: A Program Manager's Guide.
Technical Report CMU/SEI-87-TR-9, Software Engineering Institute, May, 1987.

[Gomaa 84] Gomaa, H.
A Software Design Method for Real-Time Systems.
CACM 27(9):938-949, September, 1984.

[Gould 85] *Gould K115 Logic Analyzer User's Manual.*
Gould, Inc., 1985.

[Harbaugh 84] Harbaugh, S. and Forakis, J.
Timing Studies Using a Synthetic Whetstone Benchmark.
Ada Letters 4(2):23-34, 1984.

[Harel 87] Harel, D.
Statecharts: A Visual Formalism for Complex Systems.
Science of Computer Programming 8:231-274, 1987.

[Hook 85] Hook, A. A., et al.
User's Manual for the Prototype Ada Compiler Evaluation Capability (ACEC), Version 1.
Technical Report P-1879, Institute for Defense Analysis, October, 1985.

[INS Behavioral Specification 87]
Landherr, Stefan F. and Klein, Mark H.
Initial Navigation System Simulator Behavioral Specification.
Technical Report CMU/SEI-87-TR-33, Software Engineering Institute, October, 1987.

[INS-tldd 87] Klein, Mark H. and Landherr, Stefan F.
Initial Navigation System Simulator Program: Top-Level Design.
Technical Report CMU/SEI-87-TR-34, Software Engineering Institute, October, 1987.

[Liu 73] Liu, C.L. and Layland, J.W.
Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment.
JACM 20(1):46-61, January, 1973.

[LSI-11 86] *LSI-11 Analog System Users' Guide.*
Digital Equipment Corporation, 1986.

[MacLaren 80] MacLaren, Lee.
Evolving Toward Ada in Real-Time Systems.
In *Proceedings of the ACM-SIGplan Symposium on the Ada Programming Language*. November, 1980.

[Meyers 87] Meyers, B. Craig.
Systems Specification Document for an Inertial Navigation System.
Technical Report , Software Engineering Institute, to be published, 1987.

[Meyers-EC 87] Meyers, B. Craig.
Functional Performance Specification for an External Computer to Interface to an Inertial Navigation System Simulator.
Technical Report , Software Engineering Institute, to be published, 1987.

[Meyers-INS 87] Meyers, B. Craig.
Functional Performance Specification for an Inertial Navigation System.
Technical Report , Software Engineering Institute, to be published, 1987.

[MeyersI 87] Meyers, B. Craig and Cappellini, Andrea L.
The Use of Representation Clauses and Implementation-Dependent Features in Ada: I. Overview.
Technical Report SEI-87-TR-14, Software Engineering Institute, July, 1987.

[MeyersIIA 87] Meyers, B. Craig and Cappellini, Andrea L.
The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIA. Evaluation Questions.
Technical Report SEI-87-TR-15, Software Engineering Institute, July, 1987.

[MeyersIIB 87] Meyers, B. Craig and Cappellini, Andrea L.
The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIB. Experimental Procedures.
Technical Report SEI-87-TR-18, Software Engineering Institute, July, 1987.

[MeyersIIIA 87] Meyers, B. Craig and Cappellini, Andrea L.
The Use of Representation Clauses and Implementation-Dependent Features in Ada: IIIA. Qualitative Results for VAX Ada.
Technical Report SEI-87-TR-17, Software Engineering Institute, July, 1987.

[MeyersIVA 87] Meyers, B. Craig and Cappellini, Andrea L.
The Use of Representation Clauses and Implementation-Dependent Features in Ada: IVA. Qualitative Results for Ada/M(44).
Technical Report SEI-87-TR-19, Software Engineering Institute, July, 1987.

[Motorola 85] *MC68020 32-Bit Microprocessor User's Manual.*
2nd edition, Motorola, Inc., 1985.

[Motorola 86] *MVME133 VMEmodule 32-Bit Monoboard Microcomputer User's Manual*
1st edition, Motorola, Inc., 1986.

[Nielson 87] Nielson, K.W. and Shumate, K.
Designing Large Real-Time Systems with Ada.
CACM 30(8):703-715, August, 1987.

[Sha 87] Sha, L., Lehoczky, J.P., and Rajkumar, R.
A Schedulability Test for Rate-Monotonic Priority Assignment.
Technical Report CMU CSD, Carnegie Mellon University, July, 1987.

[Systems Designers 87]
Ada-Plus VAX/VMS MC68020 Volumes 1 & 2.
Systems Designers plc, 1987.

[TeleSoft 87] *The Telesoft Second Generation Ada Development System for VAX/VMS to Embedded MC680X0 Targets.*
TeleSoft, 1987.

[U.S. Department of Defense 83]
U.S. Department of Defense.
Reference Manual for the Ada Programming Language.
ANSI/MIL-STD 1815A, DoD, January, 1983.

[VAXELN Ada Release 86]
VAXELN Ada Version 1.1 Release Notes.
Digital Equipment Corporation, 1986.

[VAXELN Ada User's 86]
VAXELN Ada User's Manual
Digital Equipment Corporation, 1986.

[VAXELN Release 86]
VAXELN Release Notes.
Digital Equipment Corporation, 1986.

[VAXELN User's 85]
VAXELN User's Guide.
Digital Equipment Corporation, 1985.

[Verdix 87] *VADS VAX/VMS Motorola 68000 Family Processors, Version 5.40.*
Verdix Corporation, 1987.

[Weicker 84] Weicker, Reinhold P.
Dhrystone: A Synthetic Systems Programming Benchmark.
Communications of the ACM 27(10):1013-1030, October, 1984.

[Weiderman 87a] Weiderman, N.H., Borger, M.W., Cappellini, A.L., Dart, S.A., Klein, M.H., and Landherr, S.F.
Ada for Embedded Systems: Issues and Questions.
Technical Report CMU/SEI-87-TR-26, Software Engineering Institute, December, 1987.

[Weiderman 87b] Weiderman, N.H.
Criteria for Constructing and Using an Ada Embedded System Testbed.
Technical Report SEI-87-TR-30, Software Engineering Institute, November, 1987.

Table of Contents

List of Figures

List of Tables