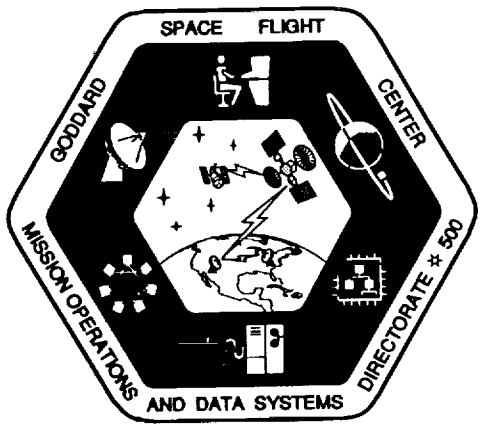
Data Systems Technology Laboratory Series *DSTL-88-001*

Ada® Projects at NASA

Runtime Environment

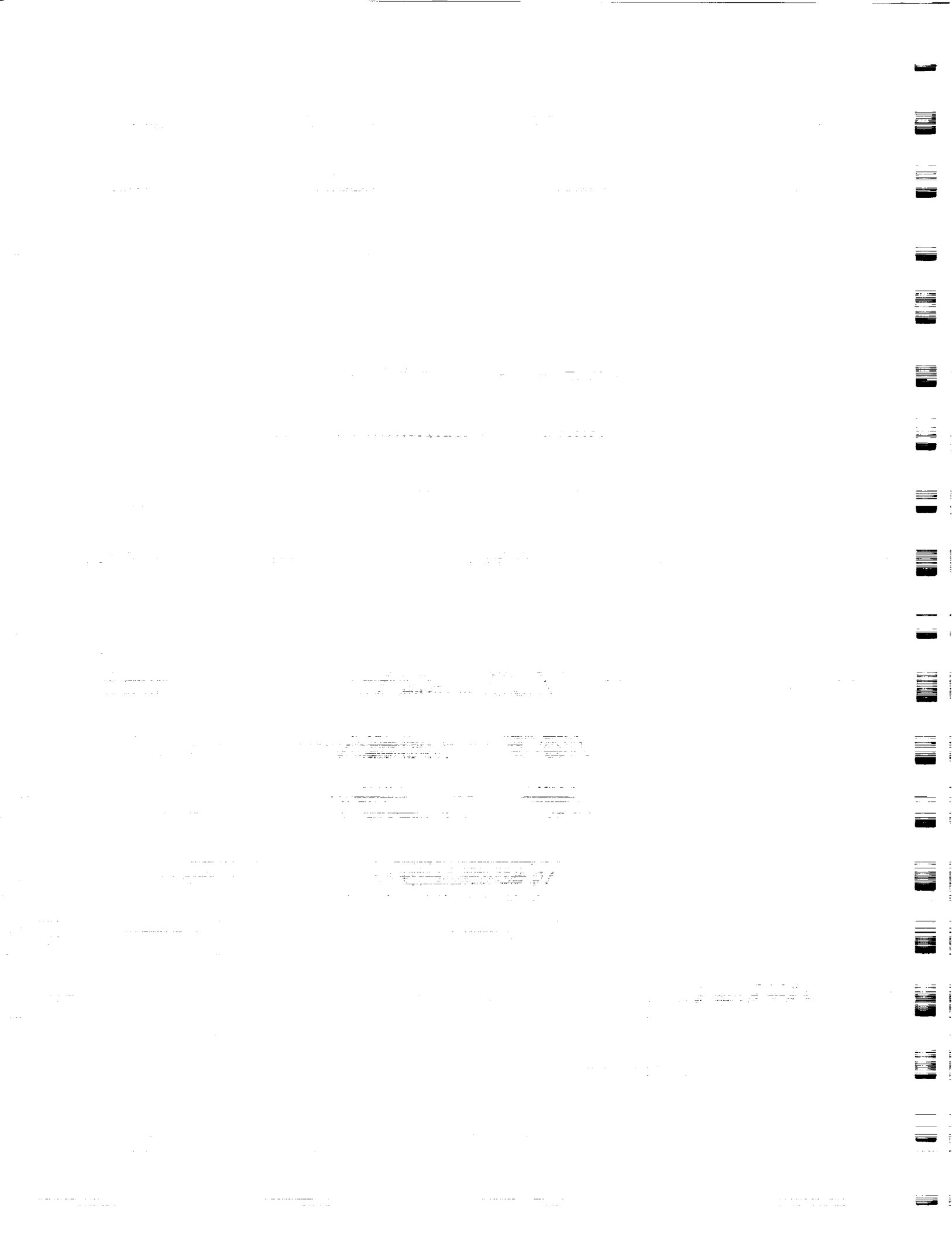
Issues and Recommendations

January 1988



NASA
National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771



FOREWORD

Ada® Projects at NASA is a publication of the Data Systems Technology Division of the Mission Operations and Data Systems Directorate, the National Aeronautics and Space Administration, Goddard Space Flight Center (NASA/GSFC).

This document is a companion document to **Ada® Runtime Packages** DSTL-88-002.

The principal authors of this document are

Daniel M. Roy

and
Randall W. Wilke
of
Century Computing, Incorporated
1100 West Street
Laurel, Maryland 20771
(301) 953-3333

Work was accomplished for

Data Systems Technology Division

under
contracts NAS 5-30017 and NAS 5-27772

Single copies of this document may be obtained by writing to

Ms. Dorothy Perkins
Code 522
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771

Ada is a registered trademark of the U.S. Government Ada Joint Program Office.

CONTENTS

SECTION 1

INTRODUCTION

1.1 INTENDED AUDIENCE

This document is intended for use by Ada practitioners to discuss and establish common short term requirements for Ada runtime environments.

The description of the Ada projects at NASA will give managers some insight into the current state of Ada.

A good knowledge of Ada and practical experience with at least one validated Ada compiler is assumed. However, such knowledge is not required to understand the executive summary and the conclusion.

1.2 PURPOSE AND SCOPE OF THE REPORT

This document is written to foster the sharing of experience and gather common immediate requirements for Ada runtime environments (RTE).

It identifies the major current Ada runtime environment issues through the analysis of some of the Ada efforts at NASA and other Research Centers.

The RTE characteristics of major compilers are compared; workaround and alternate runtime implementations are reviewed.

Modifications and extensions to the Ada Language Reference Manual (RM) to address some of these runtime issues are proposed.

Three classes of projects focusing on the most critical runtime features of Ada are recommended, including a range of immediately feasible full scale Ada development projects.

Finally, a list of Ada runtime features and procurement issues is proposed for consideration by the vendors, contractors and the Government (including AJPO).

This document is about issues; it does not dwell on Ada's numerous strengths. Dozens of studies before this one have established the

fact that no language does better than Ada on the problems and runtime issues raised [Rockwell-83] [IBM-85] [McDonnel-85] [Intermetrics-85] [Boeing-87].

This study shows Ada at work, exposing the issues, proposing solutions, and making recommendations for possible improvements.

Important notice

The issues raised in this document should not be interpreted as "problems" with Ada nor used as excuses to delay the introduction of the Ada technology where it is so badly needed.

Runtime issues for C, FORTRAN, Pascal and other "standard" languages (along with the various proprietary operating systems they run under) would be both more numerous and much more severe [Kamrad-87] [Brosgol-87] [Barnes-87].

We hope that this document will contribute to the more widespread use of Ada in aerospace applications.

1.3 EXECUTIVE SUMMARY

The Ada technology has matured rather quickly in the last two years. Over 100 compilers had been validated by August 1987, and this number was expected to reach 150 by early 1988.

Ada is currently at work in over 30 projects in all major Space Centers, totalling over 750,000 lines of code. A partial list of these projects includes the following:

- At the Goddard Space Flight Center (GSFC) in Maryland, the Gamma Ray Observatory (GRO) attitude dynamics simulator, a 100,000 LOC "pilot" project, is nearing completion [Godfrey-87]. The simulator models the interactions between the satellite Attitude Control System (ACS) and its space environment.

- Two follow-on systems, intended for operational use; GOES-I attitude dynamics simulator, and GOES-I telemetry simulator, are in their preliminary design phases. Their combined size is expected to be over 130,000 LOC [Tasaki-87].

- Also at GSFC, NASA's first operational application of Ada to flight software is under way. The Explorer Platform ACS will manage the satellite orientation in space. The 5,000 LOC system will run on a 1750A microprocessor chip set [Price-87].

- Ames Research Center in California is experimenting with Ada for parallel architectures [Goforth-87].

- The Charles Stark Draper Laboratory in Massachusetts is applying its Advanced Information Processing System fault tolerant distributed realtime Ada technology to the 30,000 LOC F-8 oblique wing flight software [CSDL-86].

- The Jet Propulsion Laboratory in Pasadena has made available to the aerospace community more than 100,000 LOC of reusable components for mathematics and astrodynamics applications [Klumpp-86]. Packages have been written to provide Ada programmers with the functionality of HAL/S, the language used for the Space Shuttle flight software.

- The Johnson Space Center (JSC) and the University of Houston at Clear Lake in Texas have been leading the Ada effort at NASA for years. They are now supporting a large number of projects for the Space Station Program [Humphrey-87].

- At the Kennedy Space Center in Florida, Doppler radar realtime data analysis software is being developed on a Compaq 386 using an Alsys compiler.

- The Solar Mesosphere Explorer is controlled from the Laboratory for Atmospheric and Space Physics in Colorado by 60,000 LOC of Ada software [Jouchoux-87].

- The Lewis Research Center in Ohio has developed a testbed for the power system of the Space Station in Ada.

- At the Marshall Space Flight Center in Alabama, a realtime system is being developed in Ada to strip classified data from three 192 kilobits per second telemetry streams for DoD classified payloads.

- The National Space Technology Laboratory in Mississipi has provided JSC and GSFC with an Ada Space Station payload simulator that is currently used to study remote controlled space science operations.

- Most of the above systems were started before the SSP Ada mandate really took effect. As impressive as they are, they pale by comparison with the recently awarded $141 million Space Station Software Support Environment (SSE). In fact, the above incomplete list is only a sign of the intensive Ada activities that will occur in the Space Station era.

Experience so far has been largely positive but it must be understood that even the best language cannot solve all software engineering problems by itself (see foot note).

People solve problems.

Ada is and will be an excellent tool in the hands of competent and educated software developers. It cannot and will not be a panacea compensating for inadequate methods or training, but it will be most beneficial if properly applied [Century-84].

This document addresses some very technical issues about the quality and efficiency of current Ada compilers. Users' needs stemming from experience on active NASA projects are enumerated and the issues are discussed. Recommendations are made for improvements via packages and compiler features needed for aerospace applications.

The busy executive should read this section as well as section 6, to obtain a quick understanding of where Ada is today, where it is going, and what it means for NASA's projects. Some insight can be gained by browsing through sections 2 and 3 where real Ada projects are described with the unedited comments from practitioners. The first page of each section contains important remarks that help put the contents of this document into perspective.

In our conclusions (Section 6.4), we make the case for a more aggressive and concerted effort to introduce and use Ada within all Space Centers.

DoD is now aggressively mandating Ada. Airbus, Boeing and FAA notably are actively involved with Ada. Of course, **NASA will use Ada for the Space Station** (see background section below), and a growing number of other software projects.

Clearly, **for the aerospace executive, the question today is no longer "When can Ada be used ?", but rather, "How can we use it now ?"**

One often mentioned issue is the cost of Ada tasking. This cost should be compared to the cost of equivalent, non Ada, alternatives. Under VMS for instance, replacing tasks with VMS processes communicating via QIOs to mailboxes (a common way of "doing tasking" from FORTRAN or C) would turn out much less efficient than the Ada equivalent solution, even in C. This was demonstrated at GSFC and at Marshall (see Section 4.2.4).

1.4 BACKGROUND

In July of 1985, the level C Space Station Project Office at the Johnson Space Center approved a configuration change board request to adopt Ada as the primary language of choice for flight subsystem application programs.

In March of 1986, the commitment of NASA's Space Station Program Office to Ada was extended to all Space Station software. A stringent waiver mechanism based on a close analysis of the cost/benefit across the entire life cycle of the Space Station Program was then put into effect.

At the Joint Conference on Ada Technology sponsored by the Goddard Space Flight Center (GSFC) in March 1987, Dr. Dana Hall, manager of the Space Station Program Office, Information Systems Management Division, reaffirmed the commitment to Ada in no uncertain terms: **"The Space Station program has selected Ada for ALL software paid for by Space Station funds. The 'all software' literally means all types: application, operating system, user interface, data base management, and any other. Needless to say, waivers to the Ada mandate will not be readily granted or easy to obtain [Hall-87]."**

Since 1984, efforts have been under way at GSFC to assess the applicability of the Ada technology to several directorates.

The GSFC approach consists of [Nelson-85]:

- Acquiring and comparing Ada Software Development Environments
- Establishing training programs and providing information exchange within and outside of GSFC
- Selecting and monitoring meaningful Ada pilot projects
- Analyzing the results of the pilot projects and making recommendations

1.5 METHODOLOGY

The following approach was followed to produce this document:

1. Contact Ada users at GSFC and other NASA Centers. Analyze their projects and report on the kinds of runtime problems currently confronting the practitioners.
2. Contact other R&D Centers (private companies, Charles Stark Draper Laboratory (CSDL), Software Engineering Institute (SEI), etc.) to survey the state of the art in RunTime Environments (RTEs).

3. Survey the literature for articles and books pertaining to Ada RTE issues.

4. Contact Ada compiler vendors to get some insight into the RTE problems facing the implementers.

5. Study the runtime user's guide (when available) for several Ada compilers.

6. Contact the relevant ACM SIGAda groups such as the Performance Issues Working Group (PIWG) and the Ada RunTime Environment Working Group (ARTEWG).

7. Give this document as widespread a review as is practically feasible.

1.6 STRUCTURE OF THIS DOCUMENT

This document is comprised of 6 sections and two appendices.

- Section 1 gives the project background and defines the purpose and scope of the report. This section also acknowledges the individuals who participated in this study.

- Section 2 is an analysis of Ada projects at GSFC.

- Section 3 is an analysis of Ada projects in other Space and Research Centers.

- Section 4 enumerates the main Ada RTE issues and makes recommendations to alleviate the problems.

- Section 5 proposes some extensions and modifications to the Ada RM.

- Section 6 makes recommendations for further analysis, tests and projects.

- Appendix A is a bibliography of Ada runtime issues.

- Appendix B is a list of Ada runtime features to consider in the procurement of Ada compilers.

1.7 ACKNOWLEDGEMENT

We gladly acknowledge the following individuals' contributions to the contents of this document:

- Dr. William Agresti (Computer Sciences Corporation, Silver Spring)
- Ms. Linda Alger (Charles Stark Draper Laboratory, Inc.)
- Mr. James P. Alstad (Hughes Aircraft Company)
- Mr. David Auty (Softech Inc.)
- Dr. Sidney Bailin (Computer Technology Associates, Inc.)
- Mr. Curtiss Barett (NASA Headquarters)
- Pr. Edward Baker (Florida State University)
- Dr. Paul Baker (Computer Technology Associates, Inc.)
- Dr. J. G. P. Barnes (Alsys Inc.)
- Mr. Mitchell Bassman (Computer Sciences Corporation, Falls Church)
- Ms. Mary Biddle (Magnavox Electronic Systems Company)
- Mr. Eric Booth (Computer Sciences Corporation, Silver Spring)
- Dr. Kenneth Bowles (Telesoft Inc.)
- Ms. Faye Brian (Lockheed Missiles & Space Company Inc.)
- Mr. Ronald Brender (Digital Equipment Corp.)
- Ms. Elisabeth Brinker (NASA Goddard Space Flight Center, Code 522.1)
- Dr. Benjamin Brosgol (Alsys Inc.)
- Mr. Robert Burkhardt (Laboratory for Atmospheric and Space Physics, University of Colorado at Boulder)
- Mr. Brian Carlson (NASA Kennedy Space Center)
- Ms. Sijung Joan Chang (Computer Sciences Corporation, Silver Spring)
- Dr. George W. Cherry (Thought**Tools Inc.)
- Mr. Joel Cohen (Computer Sciences Corporation, Silver Spring)
- Mr. Robert Conti (Digital Equipment Corp.)
- Mr. Robert Converse (Computer Sciences Corporation, Falls Church)

o Dr. Robert Dewar (New York University, Courant Institute)

o Dr. Louis DiAcetis (Bronx Community College, NY)

o Ms. Audrey Dorofee (MITRE Corp.)

o Ms. Megan Dowd (Computer Sciences Corporation, Silver Spring)

o Mr. Lance Drane (Charles Stark Draper Laboratory, Inc.)

o Mr. Curtis Emerson (NASA Goddard Space Flight Center, Code 522.2)

o Dr. Edward Fallis (Alsys Inc.)

o Mr. Daniel Ferry (Computer Sciences Corporation, Silver Spring)

o Mr. Antony Gargaro (Computer Sciences Corporation, Moorestown)

o Mr. Dale Gaumer (Magnavox Electronic Systems Company)

o Ms. Susan E. Gibson (Concurrent Computer Corporation)

o Ms. Helen Gill (MITRE Corp.)

o Mr. Andrew Goforth (NASA Ames Research Center)

o Dr. John Goodenough (Software Engineering Institute)

o Mr. Steven Gorman (NASA Johnson Space Center)

o Mr. Kaveh Habibelahy (Computer Sciences Corporation, Silver Spring)

o Mr. Jerry Hengemilhe (Fairchild Space Company)

o Ms. Jan Heuser (NASA Kennedy Space Center)

o Ms. Wendy Holladay (NASA National Space Technology Laboratory)

o Mr. Albert Horn (Smith Advanced Techniques Inc.)

o Mr. Peter Hughes (NASA Goddard Space Flight Center, Code 522.1)

o Mr. Terry Humphrey (NASA Johnson Space Center)

o Dr. Jean Ichbiah (Alsys Inc.)

o Dr. Alan Jaworski (Ford Aerospace)

o Mr. Richard Kaiser (Century Computing, Inc.)

o Mr. Michael Kamrad (Honeywell Systems Research Center)

o Mr. Allan R. Klumpp (Jet Propulsion Laboratory)

o Dr. John Knight (University of Virginia)

o Dr. Jaynarayan H. Lala (Charles Stark Draper Laboratory, Inc.)

o Ms. Suzan Legrand (Softech Inc., Houston, Texas)

o Mr. Maurice Liaw (University of Houston at Clear Lake, Texas)

o Mr. David Littmann (RMS Inc., Lanham, Md.)

o Dr. Charles McKay (University of Houston at Clear Lake, Texas)

o Mr. Paul Maresca (Adasoft)

o Mr. John Maurer (MITRE Corp.)

o Mr. Vincent Megna (Charles Stark Draper Laboratory, Inc.)

o Mr. Philip Miller (Century Computing, Inc.)

o Mr. Robert Murphy (NASA Goddard Space Flight Center, Code 522.1)

o Mr. Henry Murray (NASA Goddard Space Flight Center, Code 511.2)

o Mr. Robert W. Nelson (NASA Headquarters, Code SSI)

o Mr. John Ong (NASA Goddard Space Flight Center, Code 700)

o Mr. Murt Page (OAO Inc.)

o Ms. Dorothy C. Perkins (NASA Goddard Space Flight Center, Code 522)

o Mr. William Price (Fairchild Space Company)

o Mr. Kelvin Quimby (Computer Sciences Corporation, Silver Spring)

o Dr. Roger Racine (Charles Stark Draper Laboratory, Inc.)

- Mr. Ralph Riordan (NASA Goddard Space Flight Center, Code 515)
- Mr. Michael Rissman (Software Engineering Institute)
- Mr. Patrick Rogers (University of Houston at Clear Lake, Texas)
- Mr. Oron Schmitt (NASA Johnson Space Center)
- Ms. Catherine Schubert (NASA Lewis Research Center)
- Ms. Barbara Scott (NASA Goddard Space Flight Center, Code 408)
- Mr. Edwin Seidewitz (NASA Goddard Space Flight Center, Code 554)
- Mr. Dwight Shank (Computer Sciences Corporation, Silver Spring)
- Mr. Seetharama Shastry (Concurrent Computer Corporation)
- Mr. Allyn Shell (Computer Sciences Corporation, Beltsville)
- Mr. Robert Shuler (NASA Johnson Space Center)
- Mr. William Sloan (NASA Kennedy Space Center)
- Ms. Marlyse Smith (Alsys Inc.)
- Dr. Thomas Smith (MITRE Corp.)
- Mr. David Solomon (Computer Sciences Corporation, Silver Spring)
- Mr. Jim Spiegel (Ford Aerospace)
- Mr. Jon S. Squire (Westinghouse Electric Corp.)
- Mr. Michael Stark (NASA Goddard Space Flight Center, Code 552)
- Ms. Jody Steinbacher (Jet Propulsion Laboratory)
- Ms. Peggy Stephens (Digital Equipment Corporation)
- Mr. Robert Stevens (NASA Marshall Space Flight Center)
- Mr. Larry Taormina (NASA Marshall Space Flight Center)
- Mr. Keiji Tasaki (NASA Goddard Space Flight Center, Code 552)

- Mr. Avram Tetewsky (Charles Stark Draper Laboratory, Inc.)
- Mr. Mike Thomas (NASA Johnson Space Center)
- Ms. Mary Ann Tompkins (Lockheed Missiles & Space Company, Inc.)
- Ms. Susan Voigt (Langley Research Center)
- Mr. Nelson Weiderman (Software Engineering Institute)
- Mr. Richard Wesenberg (NASA Kennedy Space Center)
- Ms. Mary Whalen (Charles Stark Draper Laboratory, Inc.)
- Mr. James Withrow (NASA Lewis Research Center)
- Mr. Stanley Woolley (Lockheed Engineering Management Services Corp.)

1.8 LIST OF ACRONYMS

- ACEC: Ada Compiler Evaluation Capability
- ACM: Association for Computing Machinery
- ACS: Ada Compilation System
- ACVC: Ada Compiler Validation Capability
- ADE: Ada Development Environment
- AI: Artificial Intelligence
- AJPO: Ada Joint Program Office
- AP: Application Processor
- APPL: Ada Program Partitioning Language
- ARC: Ames Research Center
- ARTEWG: Ada RunTime Environment Working Group
- AST: Asynchronous System Trap
- AVO: Ada Validation Office
- BPS: Bits per second
- BSC: Binary Synchronous Control
- C3: Concurrent Computers Corporation
- C3Ada: Concurrent Computers Corporation Ada compilation system
- CAUWG: Commercial Ada Users Working Group
- CCSDS: Consultative Committee on Space Data Systems
- CMU: Carnegie-Mellon University
- COSMIC: Computer Software Management and Information Center
- CPU: Central Processing Unit
- CSC: Computer Sciences Corporation
- CSTOL: Colorado Standard Test and Operations Language

- DDC: Dansk Datamatik Center
- DEC: Digital Equipment Corporation
- DG: Data General Corporation
- DMA: Direct Memory Access
- DMS: Data Management System
- DSTL: Data Systems Technology Laboratory
- FAA: Federal Aviation Administration
- FDAS: Flight Dynamics Analysis System
- FIFO: First In First Out
- FTS: Flight Telerobotic Servicer
- GC: Garbage Collection
- GKS: Graphical Kernel System
- GOAda: GOES-I Ada Dynamic Simulator
- GRO: Gamma Ray Observatory
- GRODY: Gamma Ray Observatory attitude Dynamics Simulator
- GSFC: Goddard Space Flight Center
- GTS: GOES-I Telemetry Simulator
- HRSO: High Resolution Solar Observatory
- IBM: International Business Machines
- ICE: In-Circuit Emulator
- IPL: Interrupt Priority Level
- ISR: Interrupt Service Routine
- JPL: Jet Propulsion Laboratory
- JSC: Johnson Space Center
- KLOC: Thousand of Lines Of Code
- KSC: Kennedy Space Center

- LaRC: Langley Research Center
- LASP: Laboratory for Atmospheric and Space Physics
- LeRC: Lewis Research Center
- LMC: (Ada) Language Maintenance Committee
- LMP: (Ada) Language Maintenance Panel
- LOC: Lines Of Code
- LSE: Language Sensitive Editor
- MIPS: Million of Instructions Per Second
- MIT: Massachusetts Institute of Technology
- MOS: Mission Operations System
- MSFC: Marshall Space Flight Center
- MSOCC: Multi-Satellite Operations Control Center
- MVS: Multiple Virtual Storage
- NA: Not Available
- NAS: National Airspace System
- NASA: National Aeronautics and Space Administration
- NASCOM: NASA Communications (system or standard)
- NCP: Network Control Program
- NFS: Network File System
- NOS: Network Operating System
- NSSC: NASA Standard Spacecraft Computer
- NSTL: National Space Technology Laboratory
- NYU: New York University
- OASIS: Operations And Science Instrument System
- OCC: Operations Control Center
- OMS: On-board Management System

- OS: Operating System
- PAMELA: Process Abstraction Methodology for Embedded Large Applications
- PCA: Performance Coverage Analyser
- PC/AT: Personal Computer Advanced Technology
- PCEE: Portable Common Execution Environment
- PIWG: Performance Issues Working Group
- QA: Quality Assurance
- QIO: Queued Input Output
- RM: Ada language Reference Manual
- RPC: Remote Procedure Call
- RRM: Runtime Reference Manual
- RSOC: Remote Science Operations Center
- RTE: RunTime Environment
- RTS: RunTime System
- RTL: RunTime Library
- RV: Rendezvous
- SEI: Software Engineering Institute
- SEL: Software Engineering Laboratory
- SERC: Software Engineering Research Center
- SIG: Special Interest Group
- SIGAda: Special Interest Group on Ada
- SIS: System Interface Set
- SME: Solar Mesosphere Explorer
- SPC: Software Productivity Consortium
- SQL: Standard Query Language
- SSE: Software Support Environment

- SSIS: Space Station Information System
- SSP: Space Station Program
- STOL: Standard Test and Operations Language
- STS: Space Transportation System (space shuttle)
- TCB: Task Control Block
- TCP/IP: Transport Control Protocol / Internet Protocol
- TLM: Telemetry
- TMIS: Technical and Management Information System
- UARS: Upper Atmospheric Research Satellite
- UHCL: University of Houston at Clear Lake
- VADS: Verdix Ada Development System
- VM: Virtual Machine
- Wadas: Washington Ada Symposium
- XDR: External Data Representation

SECTION 2

ADA DEVELOPMENT EFFORTS AT GSFC

This section presents some of the Ada projects that were active at the NASA Goddard Space Flight Center in the summer of 1987.

The same format is used for each project and shows the compiler(s), host computer and OS used, the size of the team and the order of magnitude of the effort:

- "LOC completed" refers to the number of Ada source lines of code expressed in total lines of text (LOT) which includes blank lines, comments, etc., or semicolons (";"), the Ada statement terminator
- "LOC projected" refers to the expected size of the software at delivery

The ratio LOT / ";" is an indication of the amount of embedded documentation but is very dependent on coding style [GSFC-1].

Ada training, prototyping, scaffolding and other support software are not included in the above figures.

The short description for each project is followed by barely edited comments from the managers and programmers involved.

Please note

> Because of the nature of the approach followed, the issues raised in Sections 2 and 3 (Ada development projects) faithfully reproduce the statements made by practitioners. **No judgement is made about the adequacy or relevancy of the issues raised.** However, our own "NOTES" have been added where appropriate.

We hope that these honest "snapshots" will provide software professionals with a feel for the real and perceived problems they may expect when, they too, start using Ada.

2.1 GRO ATTITUDE DYNAMICS SIMULATOR (CODE 550/520)

- Project name: GRO Attitude Dynamics Simulator (GRODY)
- Compiler(s) used: DEC ACS V1.1 to 1.3
- Host computer and OS: VAX 11-780, 785 and 8600 under VMS.
- LOC completed: 135,000 lines of text (LOT), over 45,000 semicolons (";")
- LOC projected: 100,000 LOT
- Team: 4 to 7 full time.
- Project description: The Gamma Ray Observatory Attitude Dynamics Simulator project (GRODY) is a typical ground software application which models the interactions between the satellite attitude control systems and its space environment.

  In August 87, the system was being integrated; the target completion date was December 87.

  A document describing the "lessons learned" on the GRODY design has been completed and published [Godfrey-87]. A similar document on the code and test phases of the GRODY project will be available from the Software Engineering Laboratory (SEL) in December 1987.

The GRODY project has already made very valuable contributions to the introduction and use of Ada at GSFC [Nelson-86].

Several runtime issues were raised by the engineers who developed the software:

1. Problems when calling FORTRAN math functions from Ada units: When high accuracy is needed (> 9 digits), which is common for flight dynamics applications, this precision requirement should also be made explicit to the FORTRAN compiler (/G_FLOAT with DEC). Otherwise, an internal representation mismatch exists which can produce unexpected results that only careful testing can reveal. This is a generic class of problems with all imported code and should be taken into consideration no matter what the imported code language is.

   A conversion package using record representation clauses has been developed by the GRODY team to convert numeric data files to G_FLOAT format.

NOTE

DEC Ada Programmers Runtime Reference Manual (RRM) documents representation differences between the F, D, G and H formats which map to Ada 6, 9, 15 and 33 digits respectively (RRM Section 3.1.3).

2. Inconsistent behavior of exceptions across implementations [Brinker-85]: Different validated compilers may raise either NUMERIC_ERROR or CONSTRAINT_ERROR exceptions under the same test conditions. These different runtime behaviors can be traced to different interpretations of the RM.

3. Tasking is inefficient: Rendezvous (RV) overhead on DEC's Ada Compilation System (ACS) was measured to reach 50 to 100 times that of the procedure call. The GRODY team avoided the problem by using tasking very sparingly: only 6 tasks were used, 5 of them for the user interface.

NOTE

In August 87, ACM benchmarks found this ratio to be about 30 for DEC ACS. See Section 4.2.4 of this document.

The classic "alternative solution" of using QIOs to mailboxes between separate VMS processes was benchmarked to be less efficient than even the first implementation of the rendezvous [Brinker-86].

Other issues were also raised by the GRODY team:

- Compiler runtime bugs seem to be more frequent with Ada than with other, more established languages. This is not only because Ada compilers are big and that the technology is still maturing, but is probably also due to the fact that Ada provides facilities that were absent in previous languages, such as tasking and dynamic memory allocation. For instance, access types may generate CONSTRAINT_ERROR with the version 1.2 of DEC ACS. The problem goes away when the code is recompiled with no optimization (version 1.1 did not have this bug).

NOTE

Ada compilers are huge, averaging 400,000 lines of Ada code. But there is no indication in the literature or otherwise that they fail more often than C or FORTRAN compilers that have had decades to mature.

Code quality for most Ada compilers is amazing considering their size. This is probably due to the re-use of the same compiler front-end by several vendors, the use of formal methods, and the self compilation method (Ada compilers compiling themselves). However, optimizer bugs can be expected.

- Ada math packages should be standardized.

NOTE

There are several efforts in this direction [Fisher-84] [Kok-84] [Squire-87] but progress has been slow. Consensus is hard to reach.

- Interface to host operating system services is required and should be provided via well documented packages.

NOTE

DEC and Concurrent Computer Corporation (Concurrent) are compliant here.

- A clearly annotated RM, describing the implementer's interpretation and decisions, as well as a detailed runtime user's guide, are invaluable documents.

NOTE

DEC ACS documentation is a model in that respect.

- There are great variations in quality across implementations: Even validated compilers can differ greatly in the efficiency of the generated code.

NOTE

Progress has been spectacular. For instance, Telesoft's second generation products are 10 times faster in compile and runtime speed than the previous versions. When Boeing compared Ada compilers for the 7J7 aircraft, some vendors boasted a 20 fold performance improvement between November 86 and May 87 [Pflug-87].

- A working symbolic debugger was found to significantly increase productivity during the post design phases.

- The GRODY team also had problems with the unnecessary recompilations that sometimes result from the blind enforcement of Ada dependency rules by the vendor. Incremental compilation environments such as the Rational system have an edge on the competition here. For extremely large systems such as the SSE and other Space Station software, the impact of excessive recompilation cannot be over-estimated.

NOTE

This is not a runtime issue but it is an important one when selecting a compiler. Good design and coding practices reduce compilation unit dependencies. The use of sub-libraries and formal source code management tools such as DEC's Code Management System (CMS) or UNIX Source Code Control System (SCCS) can also somewhat alleviate this problem.

- GRODY unit and integration testing is taking longer than anticipated. This is a surprising observation that warrants further investigation.

NOTE

Our preliminary analysis suggests that the problems are due to some misuse of nesting and generic instantiation as well as to the release 1.2 of DEC ACS:

- DEC modified several predefined units which made all previously developed units obsolete. All Ada code in all libraries and sublibraries had to be recompiled.

- The DEC debugger for V1.2 was not fully debugged yet.

- Version 1.2 of the ACS seems to be marred with new annoying bugs.

The lessons learned on GRODY are now being applied to a similar, full-production project, described next.

2.2 GOES-I ADA DYNAMICS SIMULATOR (CODE 550)

- Project name: GOES-I Ada Dynamics Simulator (GOAda)
- Compiler(s) used: DEC ACS V1.4
- Host computer and OS: VAX 11-780 under VMS.
- LOC completed: None
- LOC projected: 100,000 LOT, 30,000 ";"
- Team: 5 to 7 full time for 18 months
- Project description: This recently started project features a functionality quite similar to GRODY. However, the two simulators differ significantly by their attitude and sensors subsystems. The team expects to re-use about 30% of GRODY's code.

  Like GRODY, two parallel efforts are taking place; one in FORTRAN, the other in Ada.

  Unlike GRODY however, GOAda is a full simulator intended for operational use. GRODY Ada was a prototype and research project whereas the FORTRAN version was a production effort. With GOAda, the FORTRAN team will only produce 2 major subsystems which will be integrated into a larger system on a MicroVAX II. This time, unlike GRODY, normal launch schedule pressure will be applied on the Ada team as well. Decision to proceed with Ada was made in November 1987 on the basis of schedule and cost [Tasaki-87].

  The Ada design team (in fact an entire section at GSFC) has undergone significant advanced Ada training using GRODY for code reading and maintenance exercises.

This unique experiment exemplifies the pragmatic approach used by the engineers of the SEL to introduce new software technologies at Goddard.

2.3 GOES-I TELEMETRY SIMULATOR (CODE 550)

- Project name: GOES-I Telemetry Simulator (GTS)
- Compiler(s) used: DEC ACS V1.4
- Host computer and OS: VAX 11-780 under VMS.

- o LOC completed: None
- o LOC projected: 30,000 LOT, about 10,000 ";"
- o Team: 3 full time for 15 months
- o Project description: Prior to the launch of a spacecraft, a telemetry simulator is needed to generate telemetry data for testing the main attitude ground support system.

  The telemetry data contains attitude sensor data, actuator data, science data and other information from the spacecraft.

  The data is grouped into several hundred bytes called minor frames, and tens of minor frames are grouped into major frames. Information is different from minor frame to minor frame. The primary objective of a telemetry simulator is to model attitude sensors, encode sensor data into a bit stream and construct minor and major frames.

  The team expects to reuse some of the GRODY code on this project. As usual, metrics data will be collected on the NASA GSFC SEL standard forms and will be analyzed and published after completion of the project.

  Although the development machine for GTS is the DEC VAX, the eventual target machine is the IBM 4341 under MVS, and perhaps NAS 8060 under VM. An IBM Ada compiler has been ordered some time ago, but had not yet been delivered to the Division.

This project is another example of the full-production application of Ada at the Goddard Space Flight Center.

2.4 FLIGHT DYNAMICS ANALYSIS SYSTEM (CODE 550)

- o Project name: Flight Dynamics Analysis System (FDAS)
- o Compiler(s) used: DEC ACS V1.1 to 1.3.
- o Host computer and OS: DEC VAX 11-780 under VMS.
- o LOC completed: About 25,000 LOT, 5,000 ";"
- o LOC projected: About 10,000 ";"
- o Team: 4 persons full time.
- o Project description: FDAS is a software tool for use in flight dynamics research. Its basic purpose is to assist programmers and analysts in building, testing and evaluating

flight dynamics software by providing an integrated support system for software modification and reconfiguration. This support system includes the following:

- A common library of reusable flight dynamics software components and utility functions.
- Standardized data and software interfaces.
- A window driven interface for selecting software components and configuring these components into an executable program. (In practice, these components consist of code from object libraries for flight dynamics analysis applications).

The following problems were discussed with the development team:

NOTE

Even though the FDAS project does not use Ada tasks, the development team had significant previous experience with tasking.

1. Tasking behavior seems difficult to predict and hard to reproduce.

NOTE

The RM does not impose any particular scheduling algorithm on the implementer. Tasking introduces a new design dimension that requires extensive training, and special tools to aid testing.

2. Rendezvous overhead is too high and should not be much greater than that of a procedure call.

NOTE

One implementation (Rational) already comes very close, but there is a lot to consider [Burns-85]. See issue 7 for the NCP project and section 4.2.4 of this document for more on this subject.

3. Tasks that perform I/O may block the entire process, defeating task parallelism.

NOTE

This is true of most implementations. On the ACS, it is fortunately not the case. Only I/O to process permanent files (SYS$INPUT, SYS$OUTPUT, SYS$COMMAND) will block the entire VMS process under which all Ada tasks run, but even that can easily be defeated; see the VAX Ada Programmer Runtime Reference Manual Section 2.7. Also note however, that assignment to ADA$OUTPUT can result in delays in screen updating.

4. Automatic garbage collection is needed. The RM should require that space be reclaimed as soon as the object is inaccessible. Also, pragmas to turn off automatic garbage collection should be provided.

NOTE

Automatic garbage collection is not suitable for all applications. Pragma CONTROLLED defers garbage collection until scope exit. See Section 4.2.1 of this document for a discussion of this subject.

5. A runtime traceback that also handles exception, detailing what exception was raised, where and why, is required.

6. Pragma INTERFACE should be supported by all implementations.

NOTE

Support of this important pragma, required by the RM, is not currently checked by the ACVC.

7. Operations on objects of numeric fixed types are not reliable. Nor is fixed_IO.

NOTE

Manipulation of numeric objects of a fixed point type is rather tricky. Ada style guides recommend against using fixed point. See Section 4.2.10 of this document for more on this subject.

Other issues were also raised by the FDAS team:

- When version 1.2 of DEC ACS was received, the entire Ada software had to be recompiled because DEC had modified the specs of some of the predefined packages, to add functionality.

- MIT's X-Window system should be adopted as a standard and Ada binding should be made available on all implementations.

NOTE

The Ada standardization effort is definitely going in that direction. Already Ada bindings for GKS (2-D graphics), SQL (Relational DBMS) and TCP/IP (Networking) have appeared on the market. However, it is good practice to wait for a new system to become stable before committing to it.

- There is an immediate and crucial need for common packages such as variable strings, queues and stacks, lexical analysis, parsing, etc.

NOTE

Already EVB Software Engineering is offering a broad range of such fully portable software components, but we are still far from a real software components industry.

- The integration between DEC's Language Sensitive Editor (LSE), the compiler and the debugger was very helpful during code and unit test. Some kind of test coverage tool (showing the parts of the code that have been exercised at least once) would improve the quality of the delivered code.

NOTE

On VAX VMS, DEC's Performance Coverage Analyser (PCA) seems a good candidate to fill that need.

- It would be very convenient to reenter the debugger after an application crash and find the cursor right at the crash position in the source code.

NOTE

When a crash can be expected, the program should be run with debugger. For cases where real-time conditions are important, see the DCL "SET PROCESS DUMP" command that allows

examining a postmortem dump file.

- Some implementations do not provide automatic recompilations in the order required by the RM. The equivalent of the ACS RECOMPILE command that does this should be required for all implementations.

- A tool to automatically generate body stubs from specifications would speed up testing.

NOTE

The Softech ALS had such a tool. However, most text editors would help generate the stub in no time (cut and paste or include predefined templates).

- A tool to automatically generate call trees would help testing and automate documentation.

- In the same vein, a cross reference tool that would show what entities are referenced by local units as well as where local units and objects are used would be useful.

NOTE

The ACS LINK/MAP/CROSS cross reference option is insufficient in that regard. The full map also displays internal unit names that are not documented.

- Whatever the methodology used, a tool that would generate code from diagrams and vice versa would speed up debugging and documentation.

- Ada packages supporting high resolution color graphics and non-keyboard interfaces (touch screen, mouse) should be more readily available.

NOTE

Indeed, touch screen devices seem to be popular with astronauts and are currently used on most prototypes of robotics user interfaces for the Space Station.

- Ada development requires a more potent environment than what has been customary so far. Individual 4 MIPS workstations clustered or networked to a mainframe appear like a more logical architecture than a centralized CPU system.

NOTE

Since an Ada compilation system features much more than a compiler, it needs more resources than a FORTRAN or C translator. This is not the most serious problem in these times of decreasing hardware costs and sky-rocketing software life cycle costs. Distributivity of the library however, is a difficult issue because of the compilation dependency rules.

- A pretty-printer would make QA happy while allowing the programmers to concentrate on the more important issues of design and correctness.

The August 1987 version of the FDAS software was submitted to COSMIC for general distribution. Updates and enhancements are planned for December 1987 and December 1988.

2.5 NETWORK CONTROL PROGRAM (CODE 520)

- Project name: Network Control Program (NCP)
- Compiler(s) used: DEC ACS V1.0 to 1.2
- Host computer and OS: VAX 11-780, 785 and 8600 under VMS
- LOC completed: 26,200 LOT, 7,500 ";"
- LOC projected: About 10,000 ";"
- Team: 2 persons full time, 1 half time.
- Project description: The Mission Operations and Data Systems Directorate Network (MNET) was developed by GSFC as a local area network using HYPERchannel hardware. Network Control Programs (NCPs) were being developed to use Ada (DEC ACS in particular) to implement the MNET protocol on various nodes.

  Build 1 of the NCP successfully transmitted 4096-byte blocks at approximately 300 Kbps (bits per second) using a modified X25 protocol, even though several RVs were used per block and interprocess communication was achieved via QIOs to a mailbox.

This effort was terminated when the off-the-shelf replacement system NETEX was purchased. Insofar as runtime issues are concerned, NCP probably was one of the most interesting and significant Ada efforts at GSFC because this project tackled a broad range of difficult and

crucial real-time problems:

1. Converting some of the networking software to Ada was a chore. The problem stemmed from the fact that the packet structure included components whose structure varied at runtime. The NCP team had to deal with 8 different types of records. The record type could only be determined at run time by looking at specific bits within the incoming packet. That made the strongly typed Ada variant record extremely difficult to use, at best. After experimenting with a solution involving FORTRAN, the NCP team ended up using the ACS specific "Assign_to_address" function. Incidentally, other implementations such as Alsys also feature this function.

   NOTE

   This is a serious problem that will plague many NASA telemetry systems such as those processing sub-commutated fields in minor frames. A more portable, but efficient, solution must be found. A range of possible solutions is discussed in Section 4.2.8.3 of this document.

2. A related issue deals with the pragma SUPPRESS. When Ada strong typing gets in the way, a more refined way to suppress checks such as for a particular object, should be provided.

   NOTE

   DEC ACS is at fault here. The RM defines pragma SUPPRESS for individual objects as well as types and units (RM B-14). DEC has chosen to replace SUPPRESS by a simpler implementation-specific pragma "SUPPRESS_ALL" that extends to the entire compilation unit.

   As a general rule, having to suppress checks is a suspect procedure. The use of subtypes, instead of types, should be considered (See GSFC Ada programming guidelines).

3. Pragma SUPPRESS would not work on DEC ACS. The team had to use SUPPRESS_ALL.

   NOTE

   DEC currently ignores pragma SUPPRESS. Furthermore, the DEC Runtime Reference Manual (RRM) describes the conditions under which exceptions may be raised in spite of pragma

SUPPRESS_ALL (RRM 6.1.4).

4. Problems were found when using unsigned numeric types. CONSTRAINT_ERROR would be raised when reading a file of bytes in spite of a type declaration for unsigned quantities, and the use of pragma PACK.

NOTE

This is an enduring bug in the ACS.

5. Heterogeneous networking also involves the translation of binary data. For instance, between DEC and IBM computers, byte ordering is different. Representation clauses (that control the internal representation of objects and therefore make such translation easy in Ada) should be enforced.

NOTE

Conversion of the various floating point formats between the two systems would pose still worse a problem. This is an issue for networking and distributed applications already addressed by vendors (Sun's NFS/XDR, 'C' network-to-host byte order conversion functions). Such applications will be common in the Space Station era.

Incidentally, practically none of the RM representation clauses (Chapter 13) are currently tested by the ACVC.

6. DEC ACS task control blocks would not go away even after task termination. This results in page faulting and inefficient memory usage.

NOTE

Note that this storage is reclaimed when the master terminates, or when the block is exited. Such idiosyncrasies must be clearly documented (DEC ACS is compliant here). See Section 4.2.1.3 of this document.

7. Rendezvous overhead was measured by the NCP team to reach 50 ms in the context of their application.

NOTE

This number is two orders of magnitude above what has been reported in the literature [Wilke-86], [Burger-87], [Chen-87], [PIWG-87].

The numbers published by SIGAda's Performance Issues Working Group (PIWG) are given in section 4.2.4.

Also note that for the task idioms used in practice [Cherry-84], the multi-RV overhead can be reduced by optimizations [Chen-87].

On a multi-user, virtual OS such as VMS, page faulting and resource contention can make such measurements extremely tricky.

Most importantly, the NCP team was not "benchmarking" the RV. Time slicing was ON, because of network interface hardware problems, and other users were on the system.

8. The rendezvous semantics were found insufficient. Other semantics such as remote procedure call and the send-receive semantics should be made available as well.

NOTE

"If the runtime semantics are defined precisely, the language will be criticized as preempting user or implementation choices. If there is flexibility granted to the implementation, then it will be attacked for a lack of portability. Thus, some criticism is inherent to any decision." [Brosgol-87]

9. A mechanism to share memory between Ada programs was also mentioned as necessary. This is usually done by calls to operating system dependent routines. Pragma INTERFACE and / or system specific packages are required to this end.

10. Task abort produced all sorts of strange effects on ACS. The runtime system would produce "invalid semaphore" messages on the 8th or 9th trial. It turned out to be difficult to obtain support from DEC about this bug.

NOTE

Very close support from the vendor on such problems is absolutely vital for the Space

Station project.

11. There was also a general feeling that inefficiencies could result from the strict requirements of the language. If entire blocks of data have to be copied in a rendezvous block, or because UNCHECKED_CONVERSION had to be used to satisfy the strong typing or other rules, the cost at runtime could be prohibitive.

NOTE

Purists insist that any use of unchecked_conversion is suspect and style books advise against using it. However, this language feature, defined in the RM, must be efficiently implemented. Good design and coding style should limit type conversions, explicit or unchecked. Furthermore, some compilers such as Tartan Lab's already implement UNCHECKED_CONVERSION with no overhead for data types of identical size. Compiler technology and hardware advances will continue to reduce the runtime cost of strong typing.

The NCP team also had the following problems with the language or its implementation:

- The DEC compiler complains when spec and body differ even when there is no semantic difference. For instance, the reserved word "IN" is not required in the declaration of parameters of mode IN. But if "IN" is explicitly indicated in the subroutine body and not in the spec, an error is generated.

NOTE

DEC ACS is correct. The two forms "differ in their sequence of lexical elements" (RM 6.3.1-8).

- Some problems were encountered when declaring units in generic packages. A body would be required whereas none was intended.

NOTE

"The syntax of a generic body is identical to that of a nongeneric body." (RM 12.2-1) That problem looks like an ACS bug.

After its termination, the NCP project was replaced with an even more ambitious Ada project, the "Modularized Gateway" that will allow NASA computers to communicate using any of the following protocols:

- DECNET
- TCP/IP
- X.25
- NETEX
- NASCOM

Some of the inter-protocol connections were already operational before the release of this report.

2.6 NOS EMULATOR (CODE 520)

- o Project name: Network Operating System Emulator (NOS Emulator)
- o Compiler(s) used: DEC ACS V1.3
- o Host computer and OS: VAX 8600 under VMS V4.5
- o LOC completed: 640 ";"
- o LOC projected: About 800 ";"
- o Team: 1 person half time
- o Project description: The NOS emulator package presents to a client program a standard interface to the Space Station Data Management System (DMS) testbed Network Operating System (NOS).

  The NOS protocol is in fact replaced by a TCP/IP connection and the CCSDS packets are transferred using sockets. This package is meant to be used by the NSTL payload simulator, also written in Ada.

  The system communicates with a Sun 3/160 workstation under UNIX in support of the High Resolution Solar Observatory (HRSO) telescience demonstration, which is written in C.

  The package interfaces to the Excelan EXOS 204 intelligent ethernet controller running EXOS 8043-02 TCP/IP network software in communicating to the Sun over ethernet.

  This project makes use of representation clauses,

UNCHECKED_CONVERSION, and VAX/VMS system services (QIOs and ASTs). One task was used to manage the NOS/socket interface.

The runtime issues raised overlap those of the RSOC project examined next.

2.7 REMOTE SCIENCE OPERATIONS CENTER (CODE 520)

- Project name: Remote Science Operations Center project (RSOC)
- Compiler(s) used: DEC ACS V1.2
- Host computer and OS: VAX 11-785 and 8600 under VMS V4.3
- LOC completed: About 1,000 ";"
- LOC projected: About 3,000 ";"
- Team: 2 to 3 part time spending about 4 hours a week each on the project.
- Project description: The Remote Science Operations Center project (RSOC) involves VAX computers in the Data Systems Technologies Laboratory at GSFC linked via land and satellite channels to a VAX 11/750 at Stanford University. The idea is to simulate remote operation of scientific experiments in orbit. This capability is a main requirement of the Space Station data management system [McKay-85]. The telecommunication and simulation software is written in C.

The project's team looked at an Ada alternative and uncovered the following:

1. VAX/VMS system services can be used extensively. That requires not only that the vendor supplies the needed packages ("STARLET" for DEC) but also that the documentation includes clearly explained examples; DEC's is good, but more examples would be welcome. Incidentally, a member of the team was sent to a DEC tutorial about using system services from Ada and found it worthwhile.

2. A "pure tasking" solution to a message buffering problem was found to be more efficient than the equivalent solution using C and mailboxes. See Section 4.2.4 of this document.

3. The team had various idiosyncratic problems mostly related to tasking. Task priorities were not found effective, time slicing did not always behave as expected, delays showed surprising jitter and memory allocation in tasks was a constant mystery.

NOTE

"In VAX Ada, a task is executed either until it becomes suspended or until a task of higher priority becomes eligible for execution" (DEC RM 9.8a).

DEC ACS offers preemptive scheduling but supports FIFO scheduling (tasks run until suspended) for tasks of the same priority to reduce context switch [Conti-87]. Pragma TIME_SLICE (round-robin) which limits the continuous execution time of tasks is used when fairness is the main requirement. See Sections 4.2.3.1..4 of this document.

4. Memory allocation for tasks is fixed and occurs at task elaboration time.

NOTE

On DEC ACS, a fixed amount of storage for stack space is allocated at task activation time [Conti-87]. Static allocation schemes for single tasks (RM 9.1-2) would reduce elaboration time.

The RM allows dynamically expanded stack space, and ACS dynamically expands the main program space (DEC RRM 7.2.2). This solution is well suited to real-time applications that could not tolerate dynamic allocation of task space.

The RSOC team also encountered the following problems, ACS bugs or design deficiencies:

1. System services do not always return the correct status code when the debugger is used.

2. Screen management routines and tasks interfere and do not work very efficiently together. Concurrent terminal I/O and processing was found difficult to achieve.

NOTE

Package TASKING_SERVICES and the use of pragma AST_ENTRY were later used. These are possible but non-portable solutions to VAX VMS specific problems.

At the time of this writing, the RSOC project was into its second phase and its PAMELA (TM) design was continuing using the Adagraph (TM) tool and code generator.

2.8 ADA PACKAGES FOR COMPUTER ACCESS TO COORDINATE REFERENCED DATA (CODE 520)

- Project name: Ada Packages for Computer Access to Coordinate Referenced Data (Code 520)
- Compiler(s) used: VADS 5.1
- Host computer and OS: Sun 2/120 under Sun Rel. 2
- LOC completed: 18,400 ";"
- LOC projected: 20,000 ";"
- Team: 1 full time, 1 half time
- Project description: The main goals of this project were to design and implement a set of packages that would:
  - Capture the differences and the relationships between various spatial coordinate systems
  - Implement an efficient index method for spatial coordinates
  - Provide an Ada interface to a relational data base
  - Generate screen menus control structure from Ada specification

  The full completion of this project was hampered by problems with the compilation system.

2.9 MULTI SATELLITE OPERATIONS CONTROL CENTER (CODE 510)

The Multi Satellite Operations Control Center (MSOCC) has been sponsoring Ada activities since 1984. From the start, the studies and projects have been centered on the specific requirements of typical OCCs:

1. A pilot project simulating most of the functionality of an OCC

2. An Ada Compilers Benchmark Suite

2.9.1 MSOCC Ada Pilot Project

- Project name: MSOCC Ada Pilot Project
- Compiler(s) used: DEC ACS V1.0
- Host computer and OS: VAX 11-780, 785 and 8600 under VMS.
- LOC completed: 4,500 LOT, 1,200 ";"
- LOC projected: 4,500 LOT, 1,200 ";"
- Team: 2 persons, 1 full time (wrote 95% of code)
- Project description: MSOCC's Ada pilot project is an Ada implementation of the Application Processor (AP) benchmark that was written to compare hardware architectures for OCC applications.

  The program simulates the following subset of AP functions:

  1. Inputs a telemetry data stream from tape at a selected rate.
  2. Decommutates the TLM data.
  3. Performs some limit checking on the data.
  4. Displays some of the TLM data on the CRT screens.
  5. Simulates the history and attitude data recording process.
  6. Simulates strip chart recorders and associated functions.
  7. Gathers statistics on the above process and generates reports.

2.9.2 MSOCC Ada Compilers Benchmark Suite

- Project name: MSOCC Ada Compilers Benchmark Suite

- o Compiler(s) used: DEC ACS, DG ADE, PC/AT, Concurrent 3200 MPS, 3260, 3280.
- o Host computer and OS: MicroVax II, VAX 11-750, 780, 785, 8250, 8600 under VMS; DG MV-4000 under AOS/VS; PC/AT under MS-DOS 3.2; Concurrent under OS/32.
- o LOC completed: 7,000 LOT, About 2,000 ";"
- o LOC projected: 7,000 LOT, About 2,000 ";"
- o Team: 2 half time.
- o Project description: The suite consists of support packages and about 100 test programs that help assess:
  1. The clarity and relevancy of the most common error messages.
  2. The compile speed for the main Ada constructs.
  3. The quality of the generated code, by inspection.
  4. The efficiency of the runtime system for the Ada constructs important for MSOCC class of applications such as tasking, dynamic memory allocation, I/O.
  5. The size of the executable modules.

The suite has since been ported to several machines and is currently being used to evaluate a beta test version of an Ada compiler for the branch's primary computers: Concurrent's 3200 MPS under OS-32.

The following runtime problems have been studied:

1. Tasking overhead: It seems that two classes of interprocess communications must be considered. The "heavy" kind involves distinct programs running on the same or different CPUs. The "light" kind involves usual Ada tasking. Since "heavy" tasking will remain the realm of the OS specific functions for a while, "pure" Ada tasking has been applied by using a combination of proven techniques and innovative methodology (PAMELA). It seems that the current rendezvous overhead can be managed with careful design for telemetry rates up to 16 kbps on an 11/785. Above that, more potent hardware and buffering of multiple blocks would be necessary to avoid loss of data.
2. Performance predictions require a good knowledge of the individual cost of the main Ada constructs. The benchmarking suite built produced useful results and allowed comparison of several implementations such as DG ADE, DEC ACS and Alsys PC/AT, and languages such as FORTRAN and C. DEC and Alsys

Ada were found to deliver production quality code that could be incorporated in current OCCs.

3. System services to handle the functions that are underspecified, or not specified, in the RM were identified:

   - Page locking, specifically for virtual OS like VMS or AOS/VS, is required.

   - Control over or intimate knowledge of the task scheduling mechanism is required. Dynamic priorities would be nice; it should be possible to turn time slicing off.

   - Control over the I/O system is required. Direct I/O such as DMA must be possible. Control over the host file and record management system is a must. It must be possible to handle the I/O of objects of mixed types (package). I/O to mass transfer devices such as NASCOM-A channels, disks, tapes, etc., performed from tasks, should not block other tasks in the system. Distributed and fault tolerant systems will impose still further requirements on the flushing of buffers to disk for instance.

     NOTE

     Low_level_IO and the other predefined I/O packages address some of these requirements. With DEC ACS, packages STARLET and TASKING_SERVICES provide related capabilities.

   - Control over and intimate knowledge of the garbage collection mechanism is required.

   - Control over the "heavy" interprocess communication, via shared memory for instance, is very important.

4. For truly embedded applications including Ada device drivers, interrupt latency must be small (typical order of magnitude: 20 µs on an 8 Mhz 80286). This requires that the compiler generates very little code when an address representation clause is used for an entry in a device driver task.

5. Clock resolution must be clearly specified and remain in acceptable limits (typical order of magnitude: 0.1 ms).

6. Support for extensive traceback, particularly in case of exception, must be provided by the runtime system. For truly embedded applications, it must also be possible to control the inclusion of such runtime code to limit memory usage.

7. Support for precise runtime performance measurement would be very welcome. For truly embedded applications, a hardware solution is preferable and should be part of the requirement for the development system, including hardware and software tools to analyze results, i.e. INTEL's and other's In-Circuit Emulators.

8. Of particular concern because of the complexity of Ada is the impact of some "small" code or data structure changes on the performance of the whole system. For instance, could a minor modification of a private type result in an order of magnitude increase in CPU time?

NOTE

This is conceivable; early PL/1 compilers had problems like this. So far, however, nothing of the kind has been reported to us or in the literature. Nevertheless, efficiency issues deserve a larger place in Ada books.

2.10 SIMULATION OPERATIONS CENTER PROJECTS (CODE 515)

The applications developed at SOC deal with simulation on a Data General MV-4000 mini-computer using the Data General Ada Development Environment (ADE) developed by Rolm.

So far, the poor runtime performance of the ADE has hindered efforts to produce code that could be used today for SOC's time critical applications.

Recently however, a software tool that helps enforce the GSFC Ada programming guidelines and standards has been produced and distributed; the tool features excellent runtime performance.

2.10.1 Pretty Printer

- Project name: Pretty printer
- Compiler(s) used: DG (Rolm) 2.3, DEC ACS 1.3, Alsys PC AT 1.0.
- Host computer and OS: DG MV-4000 under AOS/VS, VAX 8600 under VMS 4.5, PC AT under MS-DOS 3.2.

- LOC completed: 2,500 ";" (5,000 more were written to produced the parse tables)
- LOC projected: 3,500 ";"
- Team: 1 full time
- Project description: The GSFC pretty printer is an APSE tool for reformatting Ada source code in accordance with the GSFC Ada style guide.

More ambitious is the on-going NASCOM deblocker project.

2.10.2 NASCOM Deblocker

- Project name: NASCOM Deblocker
- Compiler(s) used: DG (Rolm) 2.3
- Host computer and OS: DG MV-4000 under AOS/VS
- LOC completed: 2,000 ";"
- LOC projected: 2,000 ";"
- Team: 1 full time
- Project description: The NASCOM deblocker is a real-time project for high speed communications. The Ada program will include special device drivers for NASCOM receivers and transmitters. At the highest data rate, the receiver will generate 1,000 interrupts per second.

The main runtime issues raised so far are:

- The implementation must feature an efficient mechanism to handle interrupts.
- Package MACHINE_CODE should offer the full instruction set. It should also be possible to use Ada variables in the instruction aggregate:

  INSTRUCTION'(LDA, 1, Ada_variable_name)).

- System calls should be made available along with the system parameters and their type definition. Type ADDRESS should be used wherever the system calls reference addresses (currently a type conversion may be required).

- o A CURRENT_EXCEPTION_NAME function returning a STRING should be made standard.

- o GET and PUT should accept some control characters such as BACKSPACE and BELL.

NOTE

"The effect of input or output of control characters other than horizontal tabulation is not defined by the language" (RM 14.3-7). However, DEC ACS for instance, allows the use of Put and Get with control characters. Enumeration values of type character are also handled correctly by DEC ACS:

```
ENUM_IO_OF_CHAR.put (ASCII.NUL); -- prints 'NUL'.
```

The developer also expressed the need for tools with an emphasis on the debugger:

1. A more flexible implementation of the dependency rules is called for to avoid unnecessary recompilations. At the very least, automatic recompilation in the correct order should be provided.

2. The compiler should produce an assembly language file. This was the most important feature of the ADE for the development of the NASCOM deblocker project.

NOTE

Manual editing of the assembler file to insert privileged instructions was used as a work around since the entire instruction set was not available from package MACHINE_CODE. Such a procedure would be inappropriate in a full production environment.

3. The debugger should also allow work at the assembly level; After all, Ada allows assembly language insertion.

4. Individual control of tasks should be possible from the debugger.

5. Control on the execution of delay statements should be provided.

6. It should be possible to designate which select alternative is to be be taken.

7. It should be possible to silence or dynamically control exceptions after they have been raised.

2.11 SPACECRAFT INTEGRATION TESTING (CODE 408)

2.11.1 PC AT Experiment

Experience with the Alsys Ada compiler has been mixed. Of particular concern for real-time applications are the size of the generated code and its runtime performance, which were found to be more than twice as slow as C on a typical STOL application.

NOTE

The performance of such a system is dependent on the variable length string package used. It would be interesting to try EVB's reusable components on this application. Alsys' Ada compiler has been benchmarked against several compilers for other languages (including Borland's turbo Pascal) and often ended up on top. Furthermore, Alsys' PC compiler was recently upgraded to include a "lattice algebra" high level optimizer that significantly improves runtime performance.

Further experiments will include the ingest of telemetry blocks using Alsys Ada on a PC/AT and a specially developed interface card. Performance goals are to reach telemetry speeds of up to 64 kbps (UARS). Unfortunately, this project does not seem to have received a high priority so far.

2.11.2 Spacecraft Embedded Flight Software

- Project name: Explorer Platform Flight Software
- Compiler(s) used: DEC ACS 1.4; ACT 1750A 2.1
- Host computer and OS: MicroVax II under MicroVMS
- Target Computer and OS: MDC 281 1750A Chip set
- LOC completed: 1,413 ";"
- LOC projected: 2,500 ";"

- Team: 1 full time, 1 half time

- Project description: This exciting embedded application for the Explorer Platform free-flyer spacecraft involves two processors: an NSSC-1 and a 1750A sharing a memory block. The 1750A will be used as a co-processor of the NSSC-1 to increase the throughput of the attitude control system.

  Algorithms from the Landsat NSSC-I RATFOR "update filter application processor" will be re-coded and tested in Ada, then compiled into 1750A machine assembly language.

  An "application processor" is a task that runs under the control of the on-board computer executive. The "update filter" is part of the satellite Attitude Control System and is used to process incoming star data. The satellite attitude is controlled by reaction wheels. Gyroscopes measure the acceleration in three axes and permit short term accurate determination of the satellite movement and therefore of its attitude. After a while, however gyros data need absolute recalibration to compensate for drift. The satellite's star trackers information is compared to the on-board star catalog data using Kalman filtering to recalibrate attitude data.

This is the first operational flight application of Ada within NASA and the first embedded application of Ada at the Goddard Space Flight Center. The software will fly on the "explorer platform" scheduled for launch in 1991.

Ada was selected over Assembler, Jovial and FORTRAN because of lower risk and lower cost [Hengemihle-87].

The 1750A Ada compiler has already been selected. All compilers available were compared by sending the vendors an Ada re-design of Landsat NSSC-1 Update Filter Application Processor. Vendors were asked to compile the Ada code and return a compilation listing and a link map [Hengemihle-87].

Programming activities are to start in the summer of 87.

Even though it is too early to have runtime issues raised for this project, the study has already produced interesting results:

1. Generated code is fairly compact. Memory utilization of 2 to 4 times that of assembler (for same functionality) was observed.

   NOTE

   On large projects, this ratio is fortunately smaller since not many assembler programmers can consistently outperform an automatic translator. Furthermore, some compilers'

generated code (DDC, Tartan Lab) already rival hand coded assembler.

2. Runtime system memory utilization is surprisingly moderate, varying from 8 kbytes to 28 kbytes.

NOTE

This is truly remarkable and gives the critics something to think about. Some experts expected that any Ada RTE would take a significant portion of the address space of a 1750A [Beser-85]. Tartan Laboratories now claims a base runtime space of only 1 kbyte when tasking is not used.

3. Total memory usage for the Landsat Update Filter Processor and the Ada runtime system ranges from 34 kbytes to 64 kbytes, well within limits of the 1750A address range (128 kbytes for code and 128 kbytes for data).

4. Rendezvous overhead ranges from 100 μs to 2 ms.

NOTE

It would be interesting to compare the runtime routines involved in the extreme cases since a ratio of 20 indicates very different techniques. Incidentally, the 100 μs number (and others even better) gives the critics some more to think about. In July 87, some experts still believed that the simplest RV would always require hundreds or even thousands of instructions.

2.12 OTHER ACTIVITIES AT GSFC

There are also several activities at GSFC that in spite of their importance and merit, cannot be classified yet as full-fledged Ada projects:

1. Code 522 AI activities: The Communications Link Expert System Assistance Resource (CLEAR) is a fault-isolation expert system to be implemented in the COBE POCC. This prototype system will demonstrate the capabilities of an expert system acting as an advisor, by operating in the real-time environment of a POCC. Although the expert system is written in CLIPS (an expert system shell developed by

NASA/JSC) and 'C', it is relevant here because of the expected impact of AI requirements on Ada runtime systems in the Space Station era.

Efficiency and portability are key requirements in this kind of application (real-time AI) since pattern matching is a CPU intensive activity on standard hardware.

Since it must be possible to easily interface the AI system with the rest of the software, developing expert systems in Ada will probably quickly require that the shell itself be written in Ada.

This means that the runtime requirements of efficient and controlled garbage collection, tasking, and I/O will have to be clearly met by the vendor.

Finally, as previously indicated (4.2), it might be necessary to integrate non-AI Ada and non-Ada AI software on the same machine.

2. Code 700 (Engineering Directorate): There is no active Ada project yet, but the applicability of Ada to robotics is of particular interest to the directorate in the FTS era.

2.13 FLEXIBLE ADA SIMULATION TOOL (FAST)

Though this project was not funded by GSFC (it was paid for by Ford Aerospace and Communications Corporation IR&D), the development team has been very supportive of several Ada activities within the Center and has contributed to the advancement of Ada at Goddard.

- Project name: Flexible Ada Simulation Tool (FAST)
- Compiler(s) used: Telesoft, DEC ACS V1.2
- Host computer and OS: Intellimac 7000, VAX 11-780 and Vaxstation II GPX under VMS V4.3
- LOC completed: 50,000 LOT, 20,000 ";"
- LOC projected: 50,000 LOT, 20,000 ";"
- Team: 3 to 6 persons part time, 3 full time equivalent.
- Project description: FAST is a discrete event simulation language and tool that has rapidly evolved into a complete simulation environment.

  The Ada tool features:

- Provision for interactive maintenance of a simulation input data base
- Provision for interactive maintenance of a simulation output data base
- Interactive monitoring and control of the simulation in progress

The following issues were raised by the development team:

1. Exceptions in tasks: Debugging can be difficult when unhandled exceptions result in (silent) task termination. A traceback is necessary.

2. Control over context switch: A finer level of control over when control is switched from one task to another task of the same priority is needed.

3. Entry priorities: Although one can fake entry priority by using guards, there may be a significant penalty on the RV overhead. A more expressive and potentially more efficient solution must be found.

4. Order of task activation: The RM does not specify any particular order for task activation. Inefficiencies result when one has to force a particular order with "start-up" entries.

5. Ada terminal I/O: Ada I/O mechanisms are inadequate for common terminal operations. For instance:
   - Character echo cannot be controlled through normal TEXT_IO operations
   - Processing of variable length strings input from the keyboard is difficult
   - It is difficult to code Ada routines that asynchronously respond to keyboard input without busy wait or non-portable constructs such as DEC's ASTs.

6. Dynamic strings: The lack of a predefined variable string type is bothersome. In particular, it is not possible to perform slice operations on user-defined dynamic strings.

SECTION 3

ADA DEVELOPMENT EFFORTS IN OTHER SPACE AND RESEARCH CENTERS

The analysis of Ada runtime issues continues with a look at NASA and other major aerospace projects outside GSFC.

Comments are less numerous here than in the previous section for two reasons:

1. Discussions were carried out by phone, telemail and letters, less direct forms of contact than the personal one possible at GSFC.

2. Most issues and comments raised by practitioners overlapped those raised at Goddard.

3.1 AMES RESEARCH CENTER

- Project name: Parallel Ada Research Project
- Compiler(s) used: VADS with parallel RTE.
- Host computer and OS: Sequent 4 processor under DYNIX
- LOC completed: 1,000 LOT
- LOC projected: 5,000 to 10,000 LOT
- Team: 1 full time, 1 part time
- Project description: This research project was started in January 1987, and its implications for the Space Station Project could be significant.

  The research project is composed of two parts:

  1. Basic fact finding such as:

- Performance measurement of parallel and sequential Ada
- Tradeoff between RTE and hardware configurations
- Comparative study of various synchronization mechanisms
- Evaluation of multi-processor data communication performance
- Comparative study of Ada, C and system level primitives to synchronize processes and processors

2. Pilot projects for meaningful applications such as:

- The modeling of distributed system for the Space Station Program
- Small Expert System demonstration projects
- Performance model of GSFC's Flight Telerobotic Servicer layered architecture

Since no Ada benchmarks have ever been published for parallel architectures, Arc engineers are currently developing their own.

Findings will be published by October 1987 [Goforth-87].

3.2 CHARLES STARK DRAPER LABORATORY

3.2.1 Advanced Information Processing System

- o Project name: Advanced Information Processing System (AIPS)
- o Compiler(s) used: Telesoft, Verdix.
- o Host computer and OS: VAX 8650 under VMS
- o LOC completed: NA
- o LOC projected: NA
- o Team: NA

- Project description: The Advanced Information Processing System is a novel, distributed, fault-tolerant, system architecture for life-critical digital flight control systems.

  AIPS is an on-going proof of concept prototype project. Hardware and executive software design and implementation are progressing in parallel [CSDL-86].

  "After a detailed evaluation of six candidate languages, Ada was chosen as the language most suitable for implementing the AIPS software because of its provisions for:

  - Real-time programming
  - Error detection, handling, and containment
  - Modularity and separate compilation
  - Standardization and portability" [DeWolf-84].

  The AIPS architecture consists in a network of fault tolerant multi-processors (FTMP) [Alger-86], one or more redundant I/O networks, a mass memory, and system software to manage all resources [DeWolf-84].

  The system software provides services beyond those inherent in the Ada language definition [DeWolf-84]:

  - System services such as time and file management, function migration and communication, etc.
  - I/O network services
  - Intercomputer network services [Nagle-86]
  - Local computer services (extended local RTE)

  One of the early results of this research effort is the cyclic scheduler described in section 4.2.3.5 of this document [Whitredge-87].

  The AIPS prototype has progressed to the point where it can be used operationally. Its first application is described next.

3.2.2 F8 Oblique Wing Program

- Project name: F8 Oblique Wing Program
- Compiler(s) used: VADS 5.1
- Host computer and OS: VAX 8650 under VMS
- LOC completed: 10-15,000 LOT
- LOC projected: 20-30,000 LOT
- Team: 5 full time.
- Project description: The F8 oblique wing program is the first application of the AIPS system.

  No detail was available on this on-going avionics flight software application at the time of this writing.

3.3 FEDERAL AVIATION ADMINISTRATION NATIONAL AIRSPACE SYSTEM

A recent FAA paper study identifies the functional and performance requirements for Ada runtime systems in the Advanced Automated System (AAS) era [Becker-87].

1. The Ada scheduler should run on a selectable set of conditions such as:
   - Completion of I/O processing
   - Completion of external interrupt processing
   - Task completion
   - CPU idle
   - Task suspension due to lack of required resources

NOTE

The RM does not specify the conditions under which the scheduler is run nor which scheduling algorithm must be provided. Those implementation dependent details can be taken care of by a host dependent package at the expense of portability.

2. Future air traffic control systems will need at least 13 priority levels.

NOTE

The study seems to be content with static priorities.

3. A DELAY_UNTIL (Some_absolute_time) procedure is needed for this class of real-time applications to avoid the time jitter allowed by the delay statement.

NOTE

What is meant here is that task resumption should be considered by the scheduler at the required absolute time. Resource contention, starting with CPU, can prevent the task from physically executing at the required time.

4. A SCHEDULE_AT function must be available to schedule tasks or cancel previous such requests.

NOTE

This function is underspecified in the study.

5. A lock manager task should be provided to make shared data access control more efficient.

6. Because a large number of current AAS functions would be implemented using the rendezvous, it is important that the runtime system provides an efficient implementation. The study quantifies this requirement to be 1,000 instructions on a 3 MIPS machine.

3.4 JET PROPULSION LABORATORY

Of several Ada projects at JPL, only the trajectory shaping Rendezvous guidance system and associated packages is considered in this report.

- Project name: Trajectory shaping RV guidance
- Compiler(s) used: DEC ACS V1.2
- Host computer and OS: VAX 11-780 under VMS 4.5

- o LOC completed: About 20,000 LOT
- o LOC projected: About 100,000 LOT
- o Team: 3 persons, 1 full time.
- o Project description: A relatively recent trajectory shaping RV guidance algorithm (Battin/Vaughan) and its simulation support software are being translated from HAL/S to Ada.

  In the process, all of HAL/S built-in avionics functions and other necessary mathematical entities are being implemented in Ada. These include:

  - Math and conversion constants
  - Math functions as defined in HAL/S
  - Array functions as defined in HAL/S for one and two dimensional arrays with integer, single and double precision components
  - Character and string functions (from HAL/S and PL/I)
  - Namelist package modeled after FORTRAN namelist read/write capabilities.
  - Text I/O packages to force desired defaults for get and put procedures and format text input output
  - Linear algebra packages modeled after HAL/S that include all the vector-matrix-quaternion functions used by Shuttle and Galileo software written in HAL/S as well as other, less common subprograms
  - Astrodynamics package solving Lambert's problem (Battin/Vaughan algorithm), Kepler's problem (from Battin's book), and Clohessy-Wiltshire equations.
  - Numerical algorithms to support the above packages including a Runge/Kutta integrator
  - Miscellaneous services such as date-time, messages, etc.

  The packages and their documentation are in various stages of completion and will be used operationally at JPL and JSC.

  Already, the linear algebra package has been distributed internationally to many members of the U.S. and European Ada Numerics Working Groups and is available from COSMIC [Klumpp-86].

No runtime issues per se were raised in connection with this project but several interesting language issues are documented in [Klumpp-87]

dealing with:

- The derivation of subprograms using mixed parameters and result types
- The mutual hiding of subprograms overloaded for derived types
- Exceptions declared within a generic package specification
- The overriding of TEXT_IO default parameters.

This activity exemplifies the extensibility of Ada and represents the first effort to extend HAL/S functionality to Ada. Most importantly, this project ushers in the era of truly re-usable software components at NASA.

3.5 JOHNSON SPACE CENTER

Numerous studies are being performed at JSC and several small pilot projects have been completed (FR4's AI and Ada study [Shuler-87], code EE7's TDRSS measurements data generator). Furthermore, three significant development projects are currently active at JSC: The Ada production rule system, the Ada benchmarking suite, and the DMS test bed program.

3.5.1 Ada Production Rule System

- o Project name: Ada Production Rule System (APRS)
- o Compiler(s) used: DEC ACS, Alsys PC V1.3
- o Host computer and OS: VAX 11/780 under VMS, PC/AT under MS-DOS
- o LOC completed: About 2,000 ";"
- o LOC projected: 3,500 to 5,000 ";"
- o Team: 1 senior engineer, 3/4 time
- o Project description: A system is developed for specifying rule based expert systems directly in Ada. This involves finding convenient ways of representing rules, facts, embedded procedures, lists, etc. in Ada, as well as implementing the inference engine. An earlier version is now being reworked to use a more object oriented approach in its implementation, and to make more Ada-like structuring available to the user of the system.

The developer indicates that a real-time garbage collector is necessary for production deployment of this or any similar system. In the view of the engineer in charge, it is not even theoretically possible to manage one's own storage, because no user-provided Ada program has access to all the access types globally, especially those temporarily hidden on stacks, etc.

Therefore, an efficient, automatic garbage collector is needed.

3.5.2 Ada Benchmarking Suite

- Project name: Ada benchmarking suite
- Compiler(s) used: DEC ACS, DG ADE
- Host computer and OS: MicroVax II under VAXELN, DG Rolm under OPS-32
- LOC completed: About 4,000 LOT
- LOC projected: About 8,000 LOT
- Team: NA
- Project description: The benchmarking suite borrows from several prototypes from the public domain and concentrates on testing features that are important to flight embedded applications.

  Macroscopic tests such as DHRYSTONE and WHETSTONE, will also include a synthetic dynamic benchmark built out of parts of STS flight software translated from HAL/S code.

3.5.3 DMS Test Bed

- Project name: Data Management System (DMS) Test bed
- Compiler(s) used: DEC ACS
- Host computer and OS: MicroVax II under MicroVMS
- LOC completed: About 2,500 LOT
- LOC projected: About 6,000 LOT

- o Team: 6 part-time

- o Project description: The Data Management System Test bed is an heterogeneous network of JSC test beds with DMS interfaces. Apollo/Domain, Suns, Symbolics, INTEL, IMI, DGs and VAXes are part of the net.

  The use of the DMS test bed currently include:

  - DMS test bed Network Operating System (NOS) development
  - Subsystem/DMS familiarization
  - Subsystem/subsystem data interchange
  - Space Station Information System (SSIS)/DMS functional testing
  - Contractor's investigations

  The software will be extended to STAR BUS Gateway (to GSFC), OSI protocol development, On-board Management System concept development, Space Station Subsystems integrated simulation, SSIS end to end testing and DMS service development and testing.

The problems encountered by the development team are similar to those already described and include:

1. Need for a true Ada GKS binding; Pragma INTERFACE to FORTRAN GKS is currently used.
2. Need for representation clauses to handle file format differences during data transfer between heterogeneous nodes.

3.6 KENNEDY SPACE CENTER

3.6.1 Clear Air Wind Sensing Doppler Radar

- o Project name: Clear Air Wind Sensing Doppler Radar
- o Compiler(s) used: DEC ACS 1.3, Alsys Compaq 386 compiler.
- o Host computer and OS: VAX 785 under VMS, Compaq 386 under MS-DOS.
- o LOC completed: About 5,000 LOT

- o LOC projected: About 10,000 LOT
- o Team: 2 to 6 persons, 3 or 4 full time.
- o Project description: This project consists in the development of workstation software for the analysis of clear air Doppler radar data.

  The experiment should help determine if Doppler radar data can be used to forecast thunderstorms under otherwise quiescent conditions.

  The emphasis will be on building portable software. Upon successful completion of the prototype, the software could be ported to other real-time systems.

3.6.2 Space Station Operations Language

- o Project name: Space Station Operations Language (SSOL),
- o Compiler(s) used: DEC ACS 1.0, DEC Pascal, DEC FORTRAN, DEC C
- o Host computer and OS: VAX 780 under VMS 4.x
- o LOC completed: About 1,400 ";" of Ada, 20,000 LOC of other languages.
- o LOC projected: NA
- o Team: 10 full time. Ada programming: 1 part time.
- o Project description: The Space Station Operations Language (SSOL) prototype system is a testbed for demonstrating and evaluating a real-time command and control operations language and related user environment concepts for all phases of test and checkout operations at KSC.

  Major features of the SSOL prototype system included interactive demonstrations of the following capabilities:

  - Development of graphical displays using a "graphics workbench"
  - Migration of such displays between computer systems
  - Development, execution, and maintenance of test procedures utilizing object oriented programming concepts

- Development and maintenance of test system databases

Several different methods of user interaction with the system have been demonstrated including voice control, digitizer tablet interface, and remote diagnosis of system faults by an expert system.

Ada was used primarily as an exercise to gain familiarity with the language and its capabilities.

Follow-on development has moved to the larger Core Data System (CDS) Testbed, a multi-processor, distributed environment of AT&T's 3B2 and 3B15 under UNIX V.

Larger scale use of Ada on this project hinges on the availability of quality compilers for the above systems.

3.6.3 Ada Evaluation Using A CDS Remote Interface Module

- Project name: Ada Evaluation using a Core Data System Remote Interface Module (RIM)
- Compiler(s) used: Systems Designers Ada cross compiler
- Host computer and OS: VAX 780 under VMS 4.x
- Target computer and OS: Motorola 68010 bare board computer
- LOC completed: About 340 ";"
- LOC projected: About 540 ";"
- Team: 1 part time.
- Project description: The Core Data System (CDS) prototype is a testbed for designing and developing a common set of concepts and applications to support the various Shuttle and Space Station test and integration operations performed at KSC.

  The immediate goals of this project are:

  1. The development of real-time message handling code for the Remote Interface Module (RIM) prototype. The RIM is a subsystem based on a VME chassis with special interface cards. The RIM provide the CDS interface to the hardware being controlled.

  2. Evaluate the RIM design concept.

3. Compare Ada and C.

A parallel effort is underway to implement this system in C. The information gathered from both implementations will provide quantitative data on the performance and resource requirements of Ada as compared to C. This data will be incorporated into an Ada language evaluation study currently in progress.

3.6.4 Ground Data Management System

- Project name: Ground Data Management System (GDMS)
- Compiler(s) used: NA
- Host computer and OS: NA
- LOC completed: NA
- LOC projected: NA
- Team: NA
- Project description: The Ground Data Management System (GDMS) is a test and checkout system for use in the pre-launch element and integration testing of Space Station modules, components, and experiments.

  The GDMS is designed to be a highly distributed operating environment, offering a maximum amount of flexibility for reconfiguration to support varying test requirements. Another key design goal is to minimize the system dependencies on a specific vendor's hardware.

Early prototyping activities have identified several potentially significant concerns:

1. Ada support for distributed runtime environments.
2. Real-time response of Ada programs running under a UNIX operating system

3.6.5 User Interface Development Support System

- o Project name: User Interface Development Support System
- o Compiler(s) used: DEC ACS 1.3,
- o Host computer and OS: VAX 780 under VMS 4.x
- o LOC completed: About 1,800 ";"
- o LOC projected: About 4,500 ";"
- o Team: 1 part time
- o Project description: The User Interface Development Support System is a set of packages built upon DEC's Screen Management Guidelines (SMG) runtime library functions.

  These packages, primarily intended as a set of utilities for developing menus and transaction processing applications, provide a number of useful functions for developing "window oriented" applications targeted at character based, "VTxxx" type terminals.

  This system provides a number of generic packages for developing "pull down" menus and other concepts which provide a "Macintosh-like" environment on a character based terminal. Also provided are several procedures for entry and editing of textual and numeric data.

3.7 LABORATORY FOR ATMOSPHERIC AND SPACE PHYSICS

- o Project name: Operations And Science Instrument Support (OASIS)
- o Compiler(s) used: DEC ACS V1.0 to 1.3
- o Host computer and OS: MicroVax I to VAX 11-780 under VMS.
- o LOC completed: About 26,000 ";"
- o LOC projected: About 26,000 ";"
- o Team: 5 to 8 full time.
- o Project description: The OASIS system is an outgrowth of the experience gained by LASP (University of Colorado at Boulder) from operating the Solar Mesosphere Explorer (SME).

  In 1984, NASA asked LASP to generalize SME's mission operations System (MOS) and develop a prototype of key elements. OASIS, a software package for monitoring and

controlling a wide variety of spacecraft or space science instruments, resulted from that request [Jouchoux-87].

Since delivery, the user interface has been upgraded to support the GKS standard, a CSTOL parser is been re-written in Ada, and OASIS is being used on a variety of space projects:

- Ground testing of an instrument for UARS
- Teleoperation testbed for STS' Payload of Opportunity Carrier
- Remote instrument control center for SME

The development team's experience with Ada has been most positive.

LASP's researchers mentioned the following problems:

1. Performance problems due to intolerable RV overhead with DEC ACS 1.1 on the MicroVax I were somewhat corrected by reducing the number of tasks.

NOTE

The PAMELA methodology [Cherry-86] provides guidelines in the judicious use of tasks.

The development team measured the following RV overhead:

Computer	OS	Compiler	#entries per task	RV overhead
µVAX I	µVMS 4.1	ACS 1.1	1	7.20
µVAX I	µVMS 4.1	ACS 1.1	15	15.00
VAX 780	VMS 4.2	ACS 1.1	1	2.45
VAX 780	VMS 4.2	ACS 1.1	15	5.00

All times are in milliseconds.

NOTE

See section 4.2.4 for the PIWG numbers that reflect compiler progress since version 1.1.

2. A boolean variable was tested before having been initialized. This bug did not produce either a compiler or a runtime error.

NOTE

"The execution of a program is erroneous if it attempts to evaluate a scalar variable with an undefined value" (RM 3.2.1-18).

Preferably, the compiler should generate an error message (a warning is insufficient) or the runtime system should raise CONSTRAINT_ERROR in such cases.

3. Minor bugs were encountered. In particular, no error was generated when assigning or comparing strings of incorrect lengths.

NOTE

The assignment should raise CONSTRAINT_ERROR (fixed in ACS 1.3), but the comparison should not: "No exception is ever raised by a predefined relational operator ..." (RM 4.5.2-12). However, LASP's code fragment is interesting (Style issues are irrelevant here):

```
ERROR_CODE : STRING (1..8);
...
begin
...
if ERROR_CODE = "123456" then  -- not 8 char
```

FALSE is always returned in this case. Note that the predefined equality is defined for type STRING, which is an unconstrained array type, and therefore, valid for strings of differing lengths. Nevertheless, a warning would be welcome in the LASP example and it is probable that a compiler component such as a clever semantic analyser or Alsys's "lattice algebra" optimizer, would complain in the above case.

3.8 LANGLEY RESEARCH CENTER

- Project name: Advanced Transport Operating System (ATOPS)

- o Compiler(s) used: NA
- o Host computer and OS: NA
- o LOC completed: NA
- o LOC projected: NA
- o Team: NA
- o Project description: ATOPS is an experimental Boeing 737 navigation and control system capable of performing automatically all flight tasks from take-off through landing [Knight-87].

  The operational software is written mostly in HAL/S and this project is a theoretical analysis based on rewriting part of the software in Ada. The Ada code is not expected to become operational.

  However, the issues of fault-tolerance and distributivity raised by the research team are of the highest importance for NASA applications in the space station era.

The following issues were raised by the researchers:

1. Ada ignores the issues of distributivity and fault-tolerance, the Ada definition assumes that the machine cannot fail [Knight-84]. For instance, if 2 tasks residing on separate nodes are engaged in a rendezvous when a failure occurs (during the RV), the caller could be permanently suspended if the server was lost since the RV would never end and the caller could not distinguish this situation from slow service.

2. Even though an approach to fault tolerance transparent to the Ada program has been described [Cornhill-83], non-transparent recovery has been proven to be possible and has therefore some theoretical basis.

3. A distributed testbed containing a runtime system providing the necessary facilities for non-transparent recovery has been built and tested with various failure scenarios.

4. Distribution and failure semantics have been defined that only require a pragma:

   ```
   pragma DISTRIBUTE (Task_on_processor_X)
   ```

   In short, the failure semantics are equivalent to abort [Knight-84].

5. Failure recovery for several control functions must be provided in a few milliseconds, imposing severe real-time requirements on the fault-tolerant runtime system. The non-transparent approach used on this experiment has been shown to be practical.

3.9 LEWIS RESARCH CENTER

NASA Lewis is responsible for building the power system for the Space Station. Two interesting embedded applications are being developed in Ada at this time:

3.9.1 Ada Control And Simulation Software

- Project name: Ada Control and Simulation Software
- Compiler(s) used: DEC ACS, ALS 8086 Cross-compiler, Softech 8086 Cross-compiler.
- Host computer and OS: VAX 11-785 under VMS.
- Target Computer and OS: Intel iSBC 86/30 (Bare microprocessor board target).
- LOC completed: 500 ";"
- LOC projected: About 1,500 ";"
- Team: 2 persons full time.
- Project description: This project involves writing Ada code for both the embedded control system and the hosted simulation software for the LeRC power system test bed.

  The test bed consists of a solar array field, battery banks, load banks, and a DC distribution bus.

  The simulation software provides the test bed environment for the control software. The control software monitors the simulation software to react to various test scenarios.

The development team had the following runtime requirements for the host:

1. Need a "standard" math library

2. Ada real-time capabilities should be expanded to match those of real-time languages

NOTE

A consensus on such extensions is extremely unlikely to emerge in the life span of the Station. Specific needs can easily be met by packages.

3. The RM should include more explicit support for distributed multi-processing

NOTE

One of the (few) consensus points at the Workshop on Ada real-time issues held in Moretonhampstead May 13-15, 1987, was that Ada does not currently address the semantics and problems of distributed applications [Brosgol-87]. No language does nor can with this fast evolving technology.

The following runtime requirements were expressed for the target:

1. LOW_LEVEL_IO is needed
2. Interrupt handling is needed, with representation clause or pragma
3. Inter-process communication between different processors is needed
4. Dynamic memory allocation and automatic garbage collection are needed
5. Representation clauses for addresses and data structure layout are needed

The development team also voiced their frustration with the Softech Ada Language System (ALS) 8086 cross-compiler. Another Softech 8086 cross-compiler had been ordered on a 30-day trial basis at the time of this writing.

3.9.2 Space Station Power System Software

- Project name: Space Station Power System Software

- Compiler(s) used: DEC ACS, ALS 8086 Cross-compiler, Softech 8086 Cross-compiler.
- Host computer and OS: VAX 11-780 under VMS and Intel MDS 310 under ISIS II.
- Target Computer and OS: Intel MDS 310 under iRMX-86.
- LOC completed: About 2,200 LOT, 1,250 ";"
- LOC projected: Same
- Team: 2 persons full time.
- Project description: A test bed similar but not identical to LeRC's is located at a contractor's location in California.

  The software was independently developed by the contractor, who raised issues similar to those above.

3.10 MARSHALL SPACE FLIGHT CENTER

3.10.1 Space Station OS Study

- Project name: Space Station Operating System Study
- Compiler(s) used: DEC ACS 1.3, Alsys Sun and PC/AT compilers
- Host computer and OS: MicroVax II under MicroVMS, Sun 3/260 under UNIX 4.2 BSD, PC/AT under MS-DOS.
- LOC completed: 800 ";"
- LOC projected: 2,600 ";"
- Team: 2 persons full time
- Project description: This project is a multi-facetted effort started to evaluate and compare software development workstations to be used as testbeds for the Space Station project.

  Matrix computation, tasking, disk-IO, etc. are part of the benchmarking suite. For some of the tests, equivalent code was also written in other languages such as C, Pascal and FORTRAN.

Preliminary results seem to indicate excellent performance with the Ada compilers tested.

For instance, with DEC ACS, the FORTRAN matrix benchmark program ran in 9.7 sec. compared to 10 sec. for Ada. Using pragma SUPPRESS_ALL however, Ada code ran in 7.4 sec. High level optimizers such as Alsys's "lattice algebra" optimizer that suppress all unnecessary checks, can be expected to significantly improve performances.

Another interesting preliminary result shows that in July 1987, tasking on DEC ACS was already faster than equivalent solutions using system services. This was also observed at GSFC [Brinker-86].

Incidentally, the same team demonstrated an Ada interface to NASA's Transportable Application Executive (TAE) via a package specification using pragma INTERFACE to C.

3.10.2 Downlink Data High Speed Processing

- Project name: Downlink Data High Speed Processing
- Compiler(s) used: C3 Ada R00-01
- Host computer and OS: Concurrent Computers Corporation (formerly Perkin Elmer) 3260 under OS-32.
- LOC completed: 0
- LOC projected: About 10,000 ";"
- Team: NA
- Project description: Software to strip classified data from telemetry stream for DoD's payloads was completed in 1984 and used operationally for 2 STS missions.

  Three downlink streams have to be handled at rates of up to 192 Kilobits per second each.

  The current 10,000 lines of code FORTRAN implementation runs on a VAX 11-785 under VMS. It was decided to replace the current system with an Ada redesigned version to run on Concurrent's 3260s under the OS-32 real-time operating system.

The re-implementation in Ada was decided in order to:

- Gain real-time experience with Ada
- Take advantage of "lessons learned" with the FORTRAN system
- Utilize Ada's superior maintainability features

- Benchmark performance and productivity with Ada

The team has already decided to use a combination of rendezvous and semaphore techniques. Interface to FORTRAN will also be possible if C3 Ada optimization is judged insufficient.

3.11 NATIONAL SPACE TECHNOLOGY LABORATORY

- Project name: Space Station Payload Simulator
- Compiler(s) used: DEC ACS V1.2, DG ADE, VADS
- Host computer and OS: VAX 11-785 and 8600 under VMS 4.5, MV-2000, 4000 and 8000 under AOS/VS, Intellimac I-7000 under UNIX.
- LOC completed: NA
- LOC projected: NA
- Team: 3 persons full time.
- Project description: The NSTL payload simulator for the Station DMS is a menu-driven software package written entirely in Ada [Holladay-87].

  The purpose of the simulator is to [Woolley-87]:

  - Provide variable data loads for testing network communications on the Station DMS testbed.
  - Establish requirements for designing DMS services such as
    - Operations Management System interactions
    - End-to-end test capability interactions
    - Subsystem interactions
    - Core data acquisition
    - Resource management
  - Support the implementation of the telescience concept in Space Station payload development and operations.
  - Be used as a training tool for payload design and/or operation

- Provide NASA with information and experience in the development of software in Ada for real-time applications.

The software has been operational at JSC since August 1986 and was installed at GSFC in September 86.

3.12 SOFTWARE ENGINEERING INSTITUTE

- o Project name: Ada Embedded Systems Testbed (AEST)
- o Compiler(s) used: DEC ACS V1.3, SDS Ada Plus 2B01 (68020 cross), VADS 5.4 (68020 cross), Telesoft Telegen II 3.13 (68020 cross).
- o Host computer and OS: MicroVax II under MicroVMS 4.5
- o LOC completed: About 5,000 ";"
- o LOC projected: About 20,000 ";"
- o Team: 8 persons full time.
- o Project description: The purpose of the 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.

  Details on the specific projects using AEST were not available at the time of this writing.

A report describing embedded systems' requirements, runtime issues, development environments characteristics and compiler selection heuristics is available from SEI [Weiderman-87].

3.13 UNIVERSITY OF HOUSTON AT CLEAR LAKE

Under a cooperative agreement contract with JSC, the University of Houston at Clear Lake has coordinated over 20 investigations by local area aerospace companies and has been spearheading the Ada effort at NASA since 1983 [Humphrey-87].

More data on UHCL's projects was not made available in time for this report.

SECTION 4

RUNTIME ISSUES AND RECOMMENDATIONS

In this Section, we take a very pragmatic and short term view of the runtime issues raised by the practitioners. We concentrate on the immediate needs and common requirements for Ada runtime systems, and recommend alternate solutions conducive to the immediate adoption of Ada for the largest possible range of projects within NASA.

NOTE

> The design of Ada involved an international team of world class computer scientists. The design team's compromises were arrived at after long and arduous discussions of very difficult and subtle issues. The 12 year, unprecedented effort was subjected all along to public scrutiny and competitive pressure.
>
> The following recommendations come from several unrelated sources, were often subjected to little scrutiny, can be inconsistent or mutually exclusive, and do not always consider all the implications of a particular proposal on the language as a whole.
>
> However, the ideas proposed have merit and should be carefully evaluated (see Section 6 of this document). Therefore, the following remarks should be seen as statements of need and possible solutions to issues raised by practitioners hoping to contribute to the advancement of Ada.

Because of the nature of the projects described in the previous Sections, the emphasis is put on hosted implementations; the difficult issues of safety, interoperability, fault tolerance and distributivity are not covered in depth. These issues are addressed in a major research project at the University of Houston at Clear Lake (UHCL); see [McKay-6-87] and [McKay-7-87] for details. Ada embedded systems issues and questions are addressed in depth in a recent SEI study [Weiderman-87] that features excellent reference and annotated bilbiography Sections.

4.1 DEFINITIONS

In [ARTEWG-5], the SIGAda runtime environment working group defines a runtime environment as "The predefined routines and common programming conventions for data and code structures." In practice, a runtime system, or runtime environment, is a library of routines called by the compiled code that provide basic services at execution time. Actually, [ARTEWG-5] notes that "The job of providing the runtime environment generally has been split between executives and the programming language's translators."

Figure 4-1, adapted from the same document, shows a model of an Ada runtime environment. The compiler's Ada runtime system, the host operating system (or executive) and the computer hardware form a "virtual machine" for Ada application programs. The predefined routines and common programming conventions mentioned in the definition refer to the generated code, calling conventions enforced by the compiler, and "hooks" to the virtual machine.

Note that, in this model, the runtime system is generated by the compiler from a runtime library; only the routines necessary at runtime are included.

Since Ada was designed for embedded applications, an Ada compiler might have to generate a runtime system without the support of an underlying executive or operating system. In fact, the delay statement, Ada tasking, representation clauses, dynamic objects allocation, and other such features, blur the separation between the compiler generated runtime system and facilities usually provided by the underlying executive or operating system.

However, since embedded applications will be developed in a host environment by using specific "back ends" (code generators), or a different compiler, it is important to make a clear distinction between:

1. Hosted runtime environments where a full-fledged operating system such as UNIX, VM/MVS, VAX/VMS, etc., is expected to support and interact with the compiler's runtime system.

2. Embedded runtime environments where the compiler's runtime library must provide all or most of the necessary runtime environment. Note that "lean and mean" executives such as Hunter and Ready's VRTX, INTEL's iRMX-86 and the like, fall in the embedded category because of the limited facilities provided. Actually, these "executives on a chip" can be seen as the Silicon portion of the runtime library.

It is to the credit of Ada that both hosted and embedded modes are supported by the same language, but the runtime systems' differences are significant.

NOTE

The above simplistic distinction between hosted and embedded environments is insufficient to address the broad range of issues raised by the entire Space Station project. In a recent study of System Interface Sets (SIS), researchers of the Software Engineering Research Center (SERC) introduce a third category of environment that requires specific runtime support: the integration environment. "This environment is responsible for the test and integration plans used to interactively advance the target environment baseline with approved changes in software emanating from the host environments. This environment is also responsible for controlling interactions with the target environment to maximize safety during emergencies." [McKay-7-87].

4.2 RUNTIME ISSUES

The distinction between hosted and embedded runtime systems is important because most Ada applications (practically ALL of them, today) will run in a hosted environment notwithstanding the embedded heritage of Ada. The significant differences between the two environments impose very different requirements on the runtime system. In particular, the hosted RTE must feature a close integration with the host environment:

- Interfacing to libraries written in other languages (not only assembler) must be possible. This is particularly important now in the absence of standardized Ada math libraries and while waiting for a full-fledged industry of reusable components. For instance, it could still be important in the future to delegate advanced AI functions to an AI language.

- Calling Ada from another language could also be useful at the expense of reliability. For AI applications, it seems logical that a frame-based system might activate an Ada daemon. Also, Ada is probably the best language for building "virtual machines", the layer of software that provides programs with a unique and consistent interface set of services, independent of the underlying hardware and operating system. The VMS Toolpack virtual machine [Iles-87], for example, is currently written in Pascal and would be much simpler in Ada.

NOTE

Calling Ada from another language raises a range of runtime issues and is bound to lower the reliability of the entire system to that

of the calling language. DEC ACS offers such an interface, see DEC RRM Section 4.4.1).

- Interfacing to the host's standard packages such as file and record management systems, networking, etc., must be provided in the form of Ada package specifications.

- Interfacing to the operating system services should be provided by encapsulating packages. Idiosyncrasies between tasking and ASTs for VMS, or signals under UNIX, must be documented in the runtime user's guide.

- A hosted environment must provide ways for processes to communicate and share data, sometimes across different address spaces. These processes may or may not be all written in Ada.

Note that the above requirements can lead to portability, safety, interoperability and other problems in the long term. These problems will not be solved before a full Portable Common Execution Environment (PCEE), as mentioned in [McKay-7-87], is developed, standardized, and mandated.

To examine the runtime issues raised by the Ada projects described in Sections 2 and 3, we will use the taxonomy proposed in [ARTEWG-5].

4.2.1 Storage Management

The storage management function is responsible for the allocation and deallocation of storage at runtime.

Certainly, in a hosted environment, there must be a way to deallocate storage! A short-lived missile guidance system might spend some productive minutes without it, but a ground system cannot.

A compiler for embedded applications may provide the minimum functionality (barely conforming to the RM) because of the target's limited address space. On the other hand, a compiler for hosted applications should provide maximum functionality.

Obviously, there is no need to include the entire runtime library if it is not entirely needed, and a compiler for embedded application would not. But on a hosted environment, it may be advantageous to share the entire RTL between several executing processes. That is the case with VAX VMS, where installing the Ada RTL as a sharable image library is advised.

The following issues were raised by the development teams:

4.2.1.1 Garbage Collection:

"Automatic garbage collection is needed".

Automatic storage reclamation of dynamically allocated objects can be an extremely time-consuming activity that may occur at an unpredictable time. For this reason, the RM does not require that garbage collection be provided, but it allows it. Ada offers the user several options to manage memory space (4.8-9..12):

- Storage for a collection may be allocated from the heap by a representation clause:

```
Buf_size_in_SU : constant        -- in storage units
        := (Buffer'SIZE          -- in bits
            / SYSTEM.storage_unit);
Type Buf_ptr is access Buffer;
for Buf_ptr'STORAGE_SIZE         -- for collection
        use 1_000 * (Buf_size_in_SU + 1);
```

- Pragma CONTROLLED defers garbage collection until scope exit.

```
pragma CONTROLLED (Buf_ptr);
```

- UNCHECKED_DEALLOCATION reclaims storage for a particular object.

```
Cur_buf_ptr : Buf_ptr;
Procedure Reclaim_Buf is new     -- Object specific
        UNCHECKED_DEALLOCATION (Buffer, Buf_ptr);
....
    --* Dequeue and use buffer
....
    Reclaim_buf (Cur_buf_ptr);  -- Gone (may be)
```

  Note that storage may or may not be immediately reclaimed. Concurrent's C3Ada for instance, requires that all objects of the entire collection be deallocated before returning storage to the heap.

Other languages such as C and Pascal use predefined functions (malloc and free, new and dispose) that convey less warning by their name than UNCHECKED_DEALLOCATION. Incidentally, since they are weakly typed, C pointers are particularly dangerous (assignment of pointers to INTEGER and back and between pointers to objects of different length are allowed).

A simple solution to the storage reclamation problem, used in some versions of Pascal, consists in saving and restoring the heap pointer:

```
....            -- Code
--* Save heap pointer
MARK_STORAGE;
....            -- More code
```

```
--* Get heap pointer back to where it was
RELEASE_STORAGE;
```

This solution, however is dangerous (global unchecked deallocation) and the same result can often be achieved safely, in legal Ada, by declaring a block.

Note that although it is easy to determine when storage may be reclaimed for a static object (the type of which is not an access type), this may be impossible for dynamic objects. Static objects have a lifetime determined by their scope; "pointers" may be copied to variable of an outer scope.

Recommendations: All implementations should support some form of storage reclamation.

Embedded implementations must feature the generic unit (UN)CHECKED_DEALLOCATION (See Section 4.2.1.2 below).

For AI applications in particular, hosted implementations should offer a "GARBAGE_COLLECTION" option, via a compiler switch or configuration file to provide full garbage collection, or some more efficient but more limited form of memory management such as in Alsys's iAPX86 cross compiler (generalized pragma CONTROLLED).

NOTE

Compilation units produced with and without garbage collection may or may not be mixed. In our opinion, they should not (for RTE and compiler simplicity reasons).

4.2.1.2 UNCHECKED_DEALLOCATION

"UNCHECKED_DEALLOCATION can be misused".

As the RM clearly indicates, UNCHECKED_DEALLOCATION must be used with great care.

Current practice is to build a package to handle memory management in Ada:

```
generic
  type Info_Type is private;
  Max_Allocated : Natural;
package User_node_manager is
--|
--| Author: James P. Alstad, Hughes Aircraft Co.
--|
  type Pointer is private;
  Nul : constant Pointer;
```

```
  type Node is
    record
      Info : Info_type;
      Link : Pointer;
    end record;

  Overflow, Illegal_Nul_Reference : exception;
  Operation_On_Unallocated : exception;

  procedure Allocate (New_Node : in out Pointer);
  --| Raise: Overflow

  procedure Deallocate (Old_Node : in out Pointer);
  --| Raise: Operation_On_Unallocated

  function Node_Of (The_Pointer : Pointer) return Node;
  --| Raise: Illegal_Nul_Reference

  procedure Assign_Info (New_Info : in Info_Type;
    To : in Pointer);
  --| Raise: Illegal_Nul_Reference, Operation_On_Unallocated

  procedure Assign_Link (New_Link : in Pointer;
    To : in Pointer);
  --| Raise: Illegal_Nul_Reference, Operation_On_Unallocated

end User_Node_Manager;
```

On hosted implementations, a function "CHECKED_DEALLOCATION" could be provided that would raise an exception "UNSAFE_DEALLOCATION" when an attempt is made to deallocate storage and the access type object's reference count is greater than one (or equivalent technique).

The designers of Ada systematically rejected language features that would increase the overhead of common constructs. Clearly the reference count technique would raise the overhead of the assignment statement for objects of an access type. Nevertheless, some experts claim that reference counts (and other such techniques) would feature a tolerable overhead in nearly all cases.

Therefore, a more radical (and more logical solution) would be to make CHECKED_DEALLOCATION the default and use a pragma to achieve the effect of UNCHECKED_DEALLOCATION.

Recommendations: Evaluate the replacement of procedure UNCHECKED_DEALLOCATION by CHECKED_DEALLOCATION. Consider adding a pragma to suppress the checks in the rare cases when the overhead was demonstrably intolerable.

```
pragma SUPPRESS (DEALLOCATION_CHECK, access_type_name);
```

For compatibility reasons, in the absence of pragma SUPPRESS, procedure UNCHECKED_DEALLOCATION would be mapped to CHECKED_DEALLOCATION and a warning issued.

4.2.1.3 Storage Reclamation For Terminated Tasks

"Storage space for task control blocks might not be reclaimed".

The issue arises for agent tasks dynamically activated by the execution of an allocator with the access type declared in the outer-most scope or in a library unit.

In that case, it might be difficult and even impossible for the RTE to deallocate the task control block (TCB) after task termination. This is because the access value might have been copied and an object might still be referencing the terminated task's TCB [Burns-85].

NOTE

> This is not as bad as it looks. On DEC ACS 1.0, a typical TCB occupied less than 3 kilobytes; Concurrent's TCB takes about 1 kbyte. Theoretically, after termination, this could be reduced to a trap (4 bytes or so) to raise TASKING_ERROR.

Recommendations: Since UNCHECKED_DEALLOCATION has no effect on task objects (13.10.1-8), CHECKED_DEALLOCATION with a reference count (or similar technique) looks like an attractive solution, here again, to avoid dangling references. May be then, the deallocation function could apply to task objects.

Alternate solutions, besides declaring such tasks and their type in inner scopes such as blocks or subprograms (DEC RRM 7.2.1), include the use of a pool of reusable agents. See [Burns-85] Chapter 10.

4.2.1.4 Bit Manipulation

"There is a need to extend Boolean operators to integer objects of 8, 16, 32 bits, etc."

By allowing the Boolean operators (or, and, xor, not) to work on linear arrays of Boolean, the Ada designers were not only consistent, they signaled that this was the correct abstraction for handling groups of bits; binary fields nicely map to array slices.

However, only representation clauses (direct binary representations) or pragma PACK for arrays of Boolean, if bit packing is supported can ensure that these operators deal with consecutive bits in memory or on the hardware (control registers for instance). Incidentally, some implementations pack on a byte or word boundary only. Therefore, pragma PACK by itself does not guarantee bit packing.

Currently, packed arrays of Boolean and separate packages can be found on some implementations. The package solution is probably the easiest to implement since no pragma PACK (which applies to all record and array types) is required.

Recommendations: The ACVC should check for the availability of bit packing via pragma PACK for arrays of Boolean. In the meantime, negotiate with the vendor for pragma PACK and bit packing to be provided. The sooner we can move away from the wrong abstractions encouraged by languages like FORTRAN, that only offer integer for bit arrays or enumeration types, the better. But at the same time, the overhead should be measured and compared to that of the corresponding assembler operation.

At the very least, the vendor should supply the aforementioned package, above all if pragma INTERFACE to assembler is not provided.

4.2.1.5 Bit Manipulations From Tasks

A major runtime problem arises from the access to bit or groups of bits from separate tasks [Dewar-87]. Consider the following:

```
Task1:
  packed_array_of_Boolean(n) := true;

Task2:
  packed_array_of_Boolean(n+1) := false;
```

In accordance with RM 9.11:

1. If a task reads a shared variable, no other task must update it

2. If a task updates a shared variable, no other task must read or update it.

Local copies of the shared variables are made identical at "synchronization points" such as at the start and at the completion of the rendezvous (RV).

Pragma SHARED (9.11-9) directs the RTE to perform updates of the shared variable copies each time they are updated, but the overhead may be significant. Simplicity and efficiency considerations probably dictated the limitation that pragma SHARED applied only to scalar and access type variables, but even if this changed, the above case would still pose problems.

On large instruction set machines such as VAX, the problem might be tolerable because the bit manipulation instructions are atomic (assuming the compiler uses them, that the packed array fits on one word, etc.). But consider the same problem:

- With a RISC machine for which several distinct instructions might be necessary.

- With multi-processors: In what order would the updates be made?

- With cache memory: Should the main memory and the individual caches be updated every time on all processors?

- If tasking is implemented as separate processes on a mono-CPU machine with multiple register sets. Chances are that each stack pointer in each set points to an individual copy of the shared variable. Updating them all at once is problematic.

The problem of sharing bit arrays between multi-processors was examined by the Ada Language Maintenance Committee (LMC) and declared unsolvable.

NOTE

> The LMC (now renamed the Language Maintenance Panel) is a group of Ada experts who examine the issues raised by the user community and decide on appropriate ACVC modifications. RM clarifications or modifications are also proposed to the Ada board. Such maintenance operations are scheduled to take place every 5 years. Since the RM is dated January 1983, a revised RM is due in 1988 but this will probably not occur before 1990.

There in no easy solution to the "tasking with shared variables" problem. With the advent of distributed hardware, it might become necessary to either forbid shared variables between tasks or to provide the user with explicit control over the update. In fact, preliminary Ada featured a specific procedure to do just that.

Recommendations: Have a second look at Ada-80's update procedure:

```
generic
  type SHARED is limited private;
procedure SHARED_VARIABLE_UPDATE (X : in out SHARED);
```

4.2.1.6 Constants Stored In ROM

An interesting issue for embedded applications, related to storage management, was raised during the May 87 workshop on Ada real-time issues:

"It should be possible to specify that data structures such as constant binary trees be placed in ROM" [Brosgol-87].

Constants of an access type are legal in Ada, and tree initialization could be done in the sequence of statements that may be part of a package body.

Since ROM space is always associated with a specific range of physical addresses, using address representation clauses is a solution to this problem. The RM specifically allows this (13.5-5) for variables and constants, subprograms, packages, task unit, and entry. An implementation-specific pragma would make such mapping more concise, (Tartan Lab's LINKAGE_NAME pragma already does this) but since memory partitioning is more a linker/loader function than a language issue, a configuration file with a tool to build it seem more appropriate.

Recommendations: For embedded applications, means to control the allocation of code from library units to the target memory must be provided, preferably outside of the Ada code.

4.2.2 Exception Management

The exception management function is invoked whenever an exception is raised by the Ada program or the virtual machine.

The following issues were raised by the development teams:

4.2.2.1 Constraint And Numeric Exceptions

"Different implementations can raise NUMERIC_ERROR or CONSTRAINT_ERROR under the same test conditions".

As correctly observed at GSFC, different implementations may raise NUMERIC_ERROR or CONSTRAINT_ERROR under identical test conditions. It is a recognized language issue that the two exceptions can sometimes be indistinguishable. Consider the following:

```
INTEGER'SUCC (INTEGER'LAST)     -- constraint_error
INTEGER'LAST + 1                -- numeric_error
```

NOTE

In fact, the CONSTRAINT_ERROR exception can be raised under any one of 18 different error conditions.

Recommendation: To be on the safe side, users should not distinguish between constraint and numeric exceptions. Both choices should be part of the same alternative in the exception handler:

```
exception
  when CONSTRAINT_ERROR | NUMERIC_ERROR =>
    ...
```

A warning should be added in the RM about the danger of relying on one or the other exception. Both definitions should remain however, because the two exceptions address logically distinct classes of

problems.

4.2.2.2 Exceptions And Debugging

On hosted environments, unhandled exceptions should generate a traceback showing the call stack contents in terms of fully qualified unit and exception names. This is particularly important for tasks. DEC ACS, Alsys, Rational, Concurrent, etc., are compliant here, whereas some implementations are not.

For embedded applications, the traceback feature will often be judged as irrelevant (Who needs a traceback for a missile guidance system during flight, for instance?), or too costly in memory space. In this case, the debugger in the development environment should feature traceback runtime support.

Recommendations: On a hosted environment, including embedded application development systems, unhandled exceptions must be propagated to the RTE and produce a traceback. In particular, this applies to exceptions raised, or propagated in tasks.

Furthermore, when traceback is available, the runtime system should be very specific about the kind of error condition that resulted in the raising of the exception.

4.2.2.3 Asynchronous Task Interruption

An interesting issue was raised during the May 87 workshop on Ada real-time issues [Brosgol-87].

The FAILURE exception as defined in GREEN [Ichbiah-79] should be considered for implementation in Ada.

The issue arises because of "the lack of a feature for interrupting a task asynchronously and 'immediately' causing it to resume execution at a given control point" [Brosgol-87].

The idea behind FAILURE was to provide a facility for a task to raise an exception in another. The target task is either interrupted or put on the scheduler's ready queue. Of course, RVs for the target tasks have to be cancelled, if pending, or would result in TASKING_ERROR if RV had started.

In the words of the designers of GREEN, "raising FAILURE for another task is a drastic measure that should only be used when normal means of communication have failed... It should be used only in extreme situations, for example, to protect a task against a possible malfunction in another task or to terminate an erroneous task" [Ichbiah-79].

NOTE

FAILURE, seems to threaten runtime systems with the "abort syndrome", adding runtime code and complicating the lives of both the compiler writer (sometimes) and the programmer (always) for a rather modest increase in functionality.

Recommendations: Recommend to AJPO that alternatives to FAILURE (more appropriately renamed "ASYNCHRONOUS") be researched such as those described at the end of Section 4.2.4 (RV management).

4.2.2.4 A Note On Optimization

Exception semantics, so critical to reliability, complicate optimization [Ryer-86]. The tension between exception management and optimization techniques that could cause exceptions to be raised at unexpected places in the code (in violation of RM 11.6) must be carefully studied for all compilers procured for critical applications.

Progress in optimization techniques have been steady [Kamrad-83], [Kirch-83]. Already, with the best Ada compilers, no instruction is executed for exception handling if no exception is raised. This is in accordance with the goal of runtime efficiency for exceptions set in the rationale [Ichbiah-79] in Section 12.5.4.

4.2.3 Processor Management

The processor management function is responsible for the scheduling of Ada tasks.

The following issues were raised by the development teams:

4.2.3.1 Tasking Behavior

"Tasking behavior seems difficult to predict and hard to reproduce".

The task scheduler is purposely underspecified in the RM. Dependence on the implementation details of a particular scheduler is a breach of portability, even to the next version of the same scheduler. If a particular timing relationship is desired, it must be, and can be, expressed in Ada.

Recommendations: Train the staff, and identify one or more senior "Ada gurus". Tasking issues are new to most application programmers and a combination of software engineering indoctrination, training, tools and methodology is needed to use tasking right.

Vendors should provide full documentation of the characteristics and behavior of the scheduler. A symbolic multi-task debugger that allows full control of tasks at runtime is essential for testing multi-task programs.

Embedded implementations must feature an even more sophisticated toolset allowing real-time execution of the target code under the control of a debugger residing in the host [Weiderman-87].

4.2.3.2 Time Slicing

The RM (9.8-4) seems to call for preemptive scheduling even though interpretations vary [Maule-86], but fairness might be better served for tasks of the same priority by a time slicing or other scheduler. If real-time applications are contemplated, it must be possible to disable time slicing in order to reduce overhead.

NOTE

> Some implementations such as the ALS do not provide for preemptive scheduling and a low priority task can keep the CPU indefinitely. The RM should more clearly disallow this interpretation.

Recommendations: At least preemptive and time slicing schedulers should be made available via pragmas or linker option.

NOTE

> An ARTEWG proposal for dynamic time slice specification is discussed later.

On hosted implementations, tools to adjust the scheduler (tune the RTE) such as those provided with some operating systems would be very useful.

For embedded applications, some users might find a need to directly tailor and even totally re-write the scheduler, preferably, but not necessarily, in Ada. Of course, the implications of such an endeavor on reliability, transportability and life cycle costs should be carefully assessed first.

4.2.3.3 Static Task Priorities

The RM requirement that task priorities be static is rather surprising. Actually, the priority of a caller can be raised during an RV if the called task happens to have a higher priority. The runtime cost for dynamic priorities does not appear to be that

significant (all real-time OS do this). Such functionality is likely to be offered by implementation dependent calls or packages anyway, hampering portability.

In fact, DDC's compilers for the 80x86 family already provide an RTE procedure to address this problem:

```
with system;
package RTS_EntryPoints is
  ...
  procedure RTS_SetPriority (tv : System.TaskValue;
                             p  : System.priority);
```

Type TaskValue is derived from integer in package system and a function is supplied to obtain task unit's IDs from the RTE:

```
with system;
function GetTaskValue (taddr : System.address)
  return System.TaskValue;
```

Recommendations: Reconsider whether the RM should continue requiring that tasks be of static priorities. A standard package to dynamically control priorities should be defined and evaluated. ARTEWG has proposed such a package:

```
with TASK_IDS;  -- See Section 4.2.10 of this document
package DYNAMIC_PRIORITIES is

  type PRIORITY is <implementation-defined>;

  procedure SET_PRIORITY (OF_TASK : in TASK_IDS.TASK_ID;
                          TO      : in PRIORITY);

end DYNAMIC_PRIORITIES;
```

It must be noted however, that ARTEWG does not recommend that the semantics of priority be changed in the RM. The above package, would "... provide finer-grained distinctions between tasks of equal or undefined Ada priority. It is recommended that if this package is supported, the standard type SYSTEM.PRIORITY be defined to have a null range to avoid confusion" [ARTEWG-2].

Please refer to [ARTEWG-2] for the rationale, an alternate proposal, examples, and other detailed considerations.

NOTE

Dynamic priorities may add significant overhead to the scheduler and could make some optimizations impossible.

4.2.3.4 Pragma PRIORITY

A related problem with static priorities is that whenever the pragma parameter value is modified, a lot of code often has to be recompiled since pragma PRIORITY must be specified in the task specification.

A more logical solution could be to specify a table of tasks and their priorities either to the linker or to the runtime system in the form of a configuration file. A tool could be used to create and maintain such a file.

Recommendations: A way to change the base priorities of tasks without having to recompile a lot of code would be welcome. The APPL solution mentioned below also seems attractive for setting task priorities, and its application to this problem should be studied.

An alternate solution: Should ARTEWG's DYNAMIC_PRIORITIES package be standardized, default task priorities could be set up, using the relevant subprograms, in the initialization part of a user-written package body. This in itself would drastically limit the amout of recompilation needed when priorities are changed.

NOTE

> Other related issues such as FIFO service on entry queues, undefined choice of open alternative and priority inversion are treated in 4.2.4 (RV management).

4.2.3.5 Synchronous And Asynchronous Task Scheduling

Another interesting ARTEWG proposal concerns cyclic and asynchronous scheduling.

A cyclic scheduler has been requested for some time by the aerospace community in spite of an eloquent opposition. For instance, John Barnes mentioned that cyclic scheduling is obsolete [Barnes-87] and several papers and reports have shown the limitations of this approach, finding it inefficient [Hood-86].

However, the HAL/S process scheduling paradigm includes a cyclic scheduler. A package featuring HAL/S scheduling functionality would be welcomed by the aerospace community, and would reduce some of the resistance to the introduction of Ada in the field.

Recommendations: For embedded applications, evaluate the implementation of a scheduling package to implement synchronous and asynchronous scheduling capabilities.

```
-- Make use of other ARTEWG packages
with TASK_IDS, DYNAMIC_PRIORITIES, CALENDAR;
```

```
package SCHEDULER is

  type EVENT is private;

  type TASK_INTIATIONS is (IMMEDIATELY, AT_TIME,
    AFTER_DELAY, ON_EVENT);

  type TASK_REPETITIONS is (NONE, REPEAT_EVERY,
    REPEAT_AFTER);

  type TASK_COMPLETIONS is (NONE, UNTIL_TIME,
    WHILE_EVENT, UNTIL_EVENT);

  type INTIATION_INFO (INTIATION : TASK_INITIATIONS
                       := IMMEDIATELY) is
    record
      case INITIATION is
        when IMMEDIATELY => null;
        when AT_TIME => T : CALENDAR.TIME;
        when AFTER_DELAY => D : DURATION;
        when ON_EVENT => E : EVENT;
      end case;
    end record;

  type REPETITION_INFO (REPETITION : TASK_REPETITIONS
                        := NONE) is
    record
      case REPETITION is
        when NONE => null;
        when REPEAT_EVERY | REPEAT_AFTER => D : DURATION;
      end case;
    end record;

  -- The following type, missing in [ARTEWG-2], was inferred.
  type COMPLETION_INFO (COMPLETION : TASK_COMPLETIONS
                        := NONE) is
    record
      case COMPLETION is
        when NONE => null;
        when UNTIL_TIME => T : CALENDAR.TIME;
        when WHILE_EVENT | UNTIL_EVENT => E : EVENT;
      end case;
    end record;

  procedure SCHEDULE (SCHEDULED_TASK : in TASK_IDS.TASK_ID;
                      PRIORITY : in DYNAMIC_PRIORITIES.PRIORITY;
                      INITIATION : in INITIATION_INFO;
                      REPETITION : in REPETITION_INFO;
                      COMPLETION : in COMPLETION_INFO;
                      REPORT_OVERRUN : in BOOLEAN := FALSE);

  procedure WAIT_FOR_SCHEDULE;

  procedure DESCHEDULE (SCHEDULED_TASK : in TASK_IDS.TASK_ID);
```

```
  procedure TOGGLE (TARGET_EVENT : EVENT);

  procedure SET (TARGET_EVENT : EVENT);

  procedure RESET (TARGET_EVENT : EVENT);

  function "or" (LEFT, RIGHT : EVENT) return EVENT;

  function "and" (LEFT, RIGHT : EVENT) return EVENT;

private

  type EVENT is <implementation-defined>;

end SCHEDULER;
```

Please refer to [ARTEWG-2] for rationale, example of use (shuttle second stage guidance program), and other detailed considerations.

4.2.3.6 CMU Rate Monotonic Scheduler

At the May 87 workshop on Ada real-time issues, CMU researchers proposed an exciting alternative to the classical cyclic scheduler [Cornhill-87].

In the stabilized rate monotonic (cyclic) scheduler, tasks are assigned priorities inversely to their CPU usage in such a way that the highest priority goes to the less demanding task. Ada FIFO selection order on entry queues would have to be replaced and a solution would have to be found to the priority inversion problem (in fact, these two issues are related). Both requirements are consistent with recommendations made in Sections 4.2.4.3 .. 4.2.4.5 of this document.

Analysis shows that all cyclic deadlines can be met as long as CPU usage remains under 65 %. For tasks using variable CPU time, the worst case is assumed.

This proposal would have a significant impact on the tasking features of the language and would very probably raise the RV overhead.

However, a proof of concept is needed since the proposal addresses a range of issues that have been raised for some time in the Ada community.

Recommendations: CMU's recently introduced "rate monotonic scheduler" should be implemented and carefully evaluated by NASA as soon as possible.

4.2.3.7 User Tailored RTE

At the May 87 workshop on Ada real-time issues, some experts proposed that user defined or tailored schedulers be made available as well as a package LOW_LEVEL_TASKING to control the suspension and resumption of tasks [Brosgol-87]. There does not seem to be any consensus on these proposals, however.

4.2.3.8 A Note On APPL

Researchers at Honeywell Systems and Research Center have developed an interesting paradigm for processor management that addresses the issues of distributivity, fault tolerance, performance, dynamic binding, etc.

The Ada Program Partitioning Language (APPL) describes the distribution of Ada entities (not only tasks but subprograms, packages and objects as well [RogersP-86]), and the replication of these entities.

Being remarkably consistent with Ada, an APPL configuration specification specifies the fragmentation of the Ada program into entities, while the configuration body specifies the mapping of these program fragments onto processors or nodes on a network [Eisenhauer-86].

This partition and mapping are distinct and totally separate from the Ada program itself. This separation of concern can be compared to Ada's concept of representation clauses that map the logical Ada data structures onto the underlying physical hardware.

4.2.3.9 Nonpreemptible Sections

"Certain time-critical sections of code must be guaranteed to be executed to completion without preemption" [ARTEWG-2].

This important requirement is in violation of RM 9.8-4. It also conflicts with the needs to limit interrupt latency and reduce the timing incertitude on the delay statement. The compromise that the user must make is part of the art of real-time programming.

9.8-4 might be accommodated by reserving a special super-high priority level for tasks that cannot be preempted, or a set of static priority levels such as VMS' real-time priorities, or by limiting the proposal to tasks of the same priority, etc.

Recommendations: Evaluate ARTEWG's implementation-defined package PREEMPTION_CONTROL:

```
package PREEMPTION_CONTROL is
```

```
  procedure DISABLE_PREEMPTION;
  --| Purpose:  Processor cannot be preempted from task
  --| until ENABLE_PREEMPTION is called.
  pragma INLINE (DISABLE_PREEMPTION);

  procedure ENABLE_PREEMPTION;
  pragma INLINE (ENABLE_PREEMPTION);

  function PREEMPTIBLE return BOOLEAN;
  pragma INLINE (PREEMPTIBLE);

end PREEMPTION_CONTROL;
```

NOTE

The interaction of PREEMPTION_CONTROL and INTERRUPT_MANAGEMENT described in 4.2.7 (interrupt management) should be carefully studied and clarified.

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

4.2.3.10 Dynamic Time Slicing

Some implementations provide pragma or binder options to statically define the time slice duration. DDC compiler for the 80x86 family offers the following interface:

```
with system;
package RTS_EntryPoints is
  ...
  procedure RTS_SetTimeSlice (tv : System.TaskValue;
                              ts : Duration);
```

This solution is fairly close to the one proposed by ARTEWG.

Recommendations: Evaluate ARTEWG's package interface to the scheduler:

```
with TASK_IDS;use TASK_IDS;
package TIME_SLICING is

  subtype SLICE is DURATION <implementation-defined>;

  procedure SET_TIME_SLICE (OF_TASK : in TASK_ID;
                            TO      : in SLICE);

end TIME_SLICING;
```

This package requires package TASK_IDS described in 4.2.10 of this

document.

Please refer to [ARTEWG-2] for rationale, alternate implementation, and other detailed considerations.

NOTE

It might be useful to add a function TIME_SLICE to this package:

```
function TIME_SLICE (OF_TASK : TASK_IDS.TASK_ID) return SLICE;
```

The function could be used to smoothly tune the time sliced scheduling as follows:

```
if (DATA_OVERRUN
  and (TIME_SLICE (NEMESIS) > BASE_SLICE)) then

  SET_TIME_SLICE (OF_TASK => TASK_IDS.SELF,
                  TO      => TIME_SLICE (TASK_IDS.SELF)
                             + TIME_SLICE_DELTA);


  SET_TIME_SLICE (OF_TASK => NEMESIS,
                  TO      => TIME_SLICE (NEMESIS)
                             - TIME_SLICE_DELTA);
```

It seems that some implementations could provide the above functionality with minimum overhead.

4.2.4 Rendezvous Management

"The Rendezvous (RV) management function implements the semantics of the Ada RV concept" [ARTEWG-5].

The following issues were raised by the development teams:

1. RV overhead "RV overhead is too high".

This is not always true. In fact, on some widely used systems, it is no longer the case. The following numbers were obtained from SIGAda's Performance Issues Working Group (PIWG):

Key	Trace	P000002	T000002	T000005	T000006
R1k	43	2.6	Too low	14.3	177
8600	45a	9.9	313	327	537
3280	-	4.86	151	203	-
9750	27	3.26	320	319	1357
MV10k	40	7.45	2859	2975	5449
MuVax	X5.3	41.7	1244	1225	1976
286	-	11.5	785	770	3410
68020	-	7.4	156	214	661

All times are in microseconds.

Symbols definition

The key and trace (PIWG key) values mean the following:

- R1k (43) : Rational R-1000 using G_5_15_0 at Rational
- 8600 (45a) : DEC VAX 8600 under VMS 4.2 using ACS 1.2-15 at Lear Siegler
- 3280 : Concurrent 3280 under OS-32 using Concurrent R00-00 at Concurrent
- 9750 (27) : Gould 9750 under UTX 1.2 using Telesoft (Telegen II?) at Gould
- MV10k (40) : DG MV10000 under AOS/VS 6.02 using ADE 2.30 at Ford Aerospace & Comm.
- µVax (X5.3) : DEC MicroVax II under Micro VMS T4.3 using ACS 1.2-15 at DEC
- 286 : INTEL iSBP 286-12 bare board at 6 Mhz using Alsys V3 PC/AT hosted cross-compiler to 80x86 Bare Machine at Alsys.
- 68020 : 68020 Bare Board at 20 Mhz, one wait state, using Alsys V3 VAX hosted cross-compiler to 68020 and Alsys's RTE at Alsys.

The tests symbols mean the following:

- P000002 : Procedure call and return time. Procedure is local, has no parameter and is not inlinable.
- T000002 : Task entry call and return time. One task active, one entry, task is declared in a separate package, no parameter in RV, no select.
- T000005 : Task entry call and return time. Ten tasks active, one entry, tasks are declared in a separate package, no parameter in RV, no select.
- T000006 : Task entry call and return time. One task with ten entries in a select statement, tasks are declared in a separate package, no parameter in RV.

Consult [PIWG-87] for a complete report covering over 40 configurations and dozens of tests.

With the exception of DG MV 10000, these numbers already show quite acceptable performance, but it will take some time before a combination of compiler techniques and hardware improvements reduce tasking overhead to 5 or 10 procedure calls (Rational has it now).

In the mean time, the cost of tasking should be compared to the cost of equivalent, non Ada, alternatives. Under VMS for instance, replacing tasks with VMS processes communicating via QIOs to mailboxes (a common way of "doing tasking" from FORTRAN or C) would turn out much less efficient than the Ada equivalent solution, even in C. This was demonstrated at GSFC and at Marshall. The following summarizes the results of the experiment at GSFC:

Program type	Overhead
Ada tasks and Q pack	0.42
Ada tasks and RVs	0.70
Ada tasks call Ada Mbx	1.4
C call C Mbx	1.62
Ada task call C Mbx	3.0

All times are in milliseconds, for a VAX 8600 under VMS, with normal time sharing use.

Explanations

The program types benchmarked were the following:

- Ada tasks and Q package : Two Ada tasks exchange messages by calling procedures in a queue package.
- Ada tasks and RVs : Two Ada tasks exchange messages via rendezvous. Strings are copied, no access type is used.
- Ada tasks call Ada Mbx : Two Ada tasks exchange messages by calling Ada procedures that issue a QIO to a mailbox.
- C call C Mbx : Two separate VMS processes, source written in C, exchange messages by calling C functions that issue a QIO to a mailbox.
- Ada tasks call C Mbx : Two Ada tasks exchange messages by calling C functions that issue a QIO to a mailbox. The C functions are called by using an Ada package specification and pragma interface to C.

By using access types, the performance of the rendezvous would have probably been even better. Consult [Brinker-86] for details.

The above results show that using QIOs to mailboxes from Ada is twice

as slow as the RV. Calling C mailboxes from C is even slower and calling C mailboxes from Ada is more than 4 times slower than the RV.

NOTE

Note that the better performance of the Ada Mbx version over the C Mbx program is due to the use of package TASKING_SERVICES that make more efficient use of the CPU (DEC RRM 7.5 and A-20).

Recommendations: Nothing in the RM **mandates** anyone to use tasking. Use the rules given in Chapter 9 of the GSFC Ada Style Guide [GSFC-1] for the conditions under which tasks should be considered. Objectively benchmark and compare alternatives to tasking.

Flying RV

The flight control system for the Airbus A340 will be implemented in Ada. Simulation shows that total RV overhead remains below 10% of the 150 ms cycle time [Kamrad-87].

4.2.4.1 Avoiding The Rendezvous

Several **legal Ada** alternatives to the RV have been proposed. All have severe problems stemming from busy wait and the shared variable problem.

Since Ada tasks "...may be implemented on multi-computers, multi-processors, or with interleaved execution on a single physical processor" (RM 9-5), the sharing of variables across different address space is legal. Of course, this poses severe implementation problems. The use of shared variables between tasks in the same virtual space is not trivial either and is strongly discouraged in all style guides.

An early non-RV solution to the mutual exclusion problem, the Dekker's algorithm, can be found in [Burns-85]:

```
procedure Dekker is

  task T1;
  task T2;

  type flag is (up, down);
  flag1 : flag := down;
        -- Set by T1 to indicate it intends to enter
        -- the critical Section
  flag2 : flag := down;
        -- Set by T2 to indicate it intends to enter
        -- the critical Section
```

```
turn : integer range 1..2 := 1;
      -- Used to arbitrate between the two tasks when
      -- both wish to enter the critical Section
      -- concurrently

task body T1 is
begin
  loop
    flag1 := up;       -- Entering CS
    while flag2 = up loop
      if turn = 2 then
        -- Back off, it is T2's turn.
        flag1 := down;
        while turn = 2 loop
          null;        -- Could be delay 0.0
        end loop;
        flag1 := up;   -- try again
      end if;
    end loop;
    -- Critical Section
    turn := 2;         -- Gives T2 a chance
    flag1 := down;     -- Release entry right
  end loop;
end T1;

task body T2 is
begin
  loop
    flag2 := up;       -- Entering CS
    while flag1 = up loop
      if turn = 1 then
        -- Back off, it is T1's turn.
        flag2 := down;
        while turn = 1 loop
          null;        -- Could be delay 0.0
        end loop;
        flag2 := up;   -- try again
      end if;
    end loop;
    -- Critical Section
    turn := 1;         -- Gives T1 a chance
    flag2 := down;     -- Release entry right
  end loop;
end T2;

begin -- Dekker
  null;
end Dekker;
```

A microcoded variant of this algorithm was used in National Semiconductor's SCMP microprocessor for multi-processor bus access control. However, this rather bulky solution cannot be easily extended to more than two tasks, is plagued (as written) by busy wait

unacceptable on a mono-CPU, and depends on stict adherence to a complex protocol. It is therefore unreliable and its efficiency advantage is uncertain.

The most classic "solution" to interprocess synchronization via shared variables for multi-processors is the spin lock:

```
Task1:
    Task_2_go := true;
....

Task2:
  Busy_wait:
    loop        -- The one microsecond RV
      exit when task_2_go;
    end loop busy_wait;
    ....
```

In this simpler case, shared variables like task_2_go cannot be safely used by tasks without pragma SHARED.

Recommendations: Pragma SHARED must be enforced by the ACVC.

The user community including NASA, FAA, etc., and AJPO should entice the vendors to produce efficient runtime implementations of the rendezvous semantics.

4.2.4.2 Other Semantics

The controversial rendezvous semantics strike a reasonably good balance between ease of use and functionality. It is possible to code semaphores and monitors with Ada tasks for instance. Efficiency can be an issue, however and GREEN proposed the concept of "generic tasks" of which SIGNAL and SEMAPHORE were predefined instances for efficiency reasons [Ichbiah-79].

The remote procedure call (RPC) semantics can be handled directly by the RV using a straight entry call and placing the procedure inside the accept statement. For distributed applications, today, packages must be built to handle RPC.

The send-receive semantics require intermediary tasks as agents to immediately buffer the incoming message and perform the selective entry call to the sender task. For distributed applications, today, packages must be built to provide send-receive functionality.

Pragmas could be standardized to help the compiler generate efficient RV code. For instance, pragma SEMAPHORE, SIGNAL, and MONITOR could be used in a task to convey a more restricted RV semantics to the compiler. For instance, Burns has proposed to extend pragma INLINE to task objects to request that the task be implemented as a monitor [Burns-85]. However, this kind of "solution" is on the verge of legal

Ada and would complicate tasking semantics. Immediate needs for the efficient implementation of such mechanisms are probably handled best with packages.

It must be understood however, that low level primitives of well known limitations are no substitute for the much superior process abstraction of tasking. The use of the following packages should be limited to cases for which in line machine code insertion or interface to assembler would be the only possible alternative.

Recommendations: The SEMAPHORE functionality could be provided by a package:

```
        package SEMAPHORE is
--|
--| Description:  Semantics can be expressed in Ada
--| (from [Ichbiah-79] p. 11-8)
--|
--|     task SEMAPHORE is
--|       entry WAIT;   -- P
--|       entry SEND;   -- V
--|     end SEMAPHORE;
--|
--|     task body SEMAPHORE is
--|     begin
--|       loop
--|         accept WAIT;        -- Enter critical Section
--|         accept SEND;        -- Leave
--|       end loop;
--|     end SEMAPHORE;
--|
--| Warning:  Semaphores are low-level unstructured primitives.
--| Their use can result in deadlocks and other corruptions.
--|

          type SEMAPHORE_TYPE is limited private;

          procedure WAIT (S : in out SEMAPHORE_TYPE);

          procedure SEND (S : in out SEMAPHORE_TYPE);

        private

          type SEMAPHORE_TYPE is
            record
              SEM : NATURAL := 1;
            end record;

        end SEMAPHORE;
```

NOTE

DDC provides an intersting variation of this idea for their 80x86 compilers:

```
with system;
package RTS_EntryPoints is
  ...
  procedure RTS_P (sem : System.Semaphore);

  procedure RTS_V (sem : System.Semaphore);
```

Pragma interface to assembler are used. The type semaphore is implemented as a public record type:

```
package system is
  ...
  type Semaphore is
    record
      counter : UnsignedWord;
      first, last : TaskValue;
    end record;
```

Recommendations: The SIGNAL functionality for both intra and inter-process synchronization could be provided by packages:

```
generic -- Allows separate classes of signals
  type SIGNAL_TYPE is (<>);
        -- Discrete, preferably enumeration.
package LOCAL_SIGNALS is -- Intra-process sync.
--|
--| Purpose:  Provides synchronization services to
--| processes within the same address space
--|
--| Warning:  Signals are subject to race conditions.
--| Their use can result in deadlocks and other program
--| corruption.
--|
   Max_nr_signals : constant := <implementation-defined>;
   type SIGNAL_RANGE is range 1 .. Max_nr_signals;

   type SIGNAL_LIST is array (SIGNAL_RANGE range <>)
     of SIGNAL_TYPE;

   procedure SET (S : in SIGNAL_TYPE);

   procedure CLEAR (S : in SIGNAL_TYPE);

   function IS_SET (S : SIGNAL_TYPE) return BOOLEAN;

   function IS_ANY_SET (S : SIGNAL_LIST) return BOOLEAN;

   function ARE_ALL_SET (S : SIGNAL_LIST) return BOOLEAN;
```

```
            procedure WAIT_FOR (S : in SIGNAL_TYPE);

            procedure WAIT_FOR_ANY (S : in SIGNAL_LIST);
                  -- Wait on logical OR of signals

            procedure WAIT_FOR_ALL (S : in SIGNAL_LIST);
                  -- Wait on logical AND of signals

         end LOCAL_SIGNALS;


         package GLOBAL_SIGNALS is -- Inter-process sync.
--|
--| Purpose:  Provides synchronization services to
--| processes across different address spaces
--|
--| Warning:  Signals are subject to race conditions.  Their use
--| can result in deadlocks and other program corruption.
--|
            type SIGNAL_TYPE is <implementation-defined>;
                  -- Discrete, preferably enumeration.

            Max_nr_signals : constant := <implementation-defined>;
            type SIGNAL_RANGE is range 1 .. Max_nr_signals;

            type SIGNAL_LIST is array (SIGNAL_RANGE range <>)
              of SIGNAL_TYPE;

            procedure SET (S : in SIGNAL_TYPE);

            procedure CLEAR (S : in SIGNAL_TYPE);

            function IS_SET (S : SIGNAL_TYPE) return BOOLEAN;

            function IS_ANY_SET (S : SIGNAL_LIST) return BOOLEAN;

            function ARE_ALL_SET (S : SIGNAL_LIST) return BOOLEAN;

            procedure WAIT_FOR (S : in SIGNAL_TYPE);

            procedure WAIT_FOR_ANY (S : in SIGNAL_LIST);
                  -- Wait on logical OR of signals

            procedure WAIT_FOR_ALL (S : in SIGNAL_LIST);
                  -- Wait on logical AND of signals

         end GLOBAL_SIGNALS;
```

Recommendations: The MONITOR functionality could be provided by a package:

```
         generic -- Adapted from [Burns-86] p. 39..42
```

```
        type RESOURCE_RANGE is range <>;

      package MONITOR is
--|
--| Warning: Monitors' procedures must be called in the right
--| order.  If not, deadlocks can occur.
--|
        procedure ACQUIRE (AMOUNT : in RESOURCE_RANGE);
              -- Atomic action.  Caller is blocked until
              -- all resources are granted

        procedure RELEASE (AMOUNT : in RESOURCE_RANGE);

      end MONITOR;
```

At the same time, variations on the RV theme should be carefully considered such as:

- Pragma SOFTWARE_INTERRUPT for task entries. This pragma would give top priority to an RV on this entry. Minimum disturbance to the normal RV semantics should be achieved. The main thrust here would be to provide asynchronous, ultra-fast, task to task signaling. This is an alternative to the FAILURE exception mentioned in Section 4.2.2.3 of this document.

- Pragma ASYNCHRONOUS_RTS_TRAP for task entries. This pragma would allow a close interaction between RTE specific packages (see Processor management below) and tasks. It could be modeled on DEC's pragma AST_ENTRY.

  This idea dovetails nicely with the current ARTEWG's work toward standard Ada interface to the runtime system. See 4.2.7 (interrupt management) for ARTEWG's "fast interrupt" proposal.

  The main thrust here would be to handle some of the elaborate distributivity and fault tolerant issues raised in the literature [Knight-84].

The RV semantics is recognized as an elegant construct. However, its implementation currently suffers from [Dewar-87]:

1. The deficiencies of the validation process that puts all the emphasis on the syntax and semantics of Ada

2. Inappropriate primitives; usually those of the underlying OS which are grossly inefficient.

3. Lack of basic research. This could be changing. EUREKA, the European high technology initiative, is spending $1 Million with Alsys Inc. alone to address these issues.

In spite of its current implementation limitations, the RV is hard to

replace. Already, as it was shown above (Section 4.2.4.1), the RV performs better than the closest equivalent solution used in the past for interprocess synchronization and communication: the mailbox.

A small number of Ada experts had predicted that the typical RV overhead could be lowered to 50 microseconds on 2 MIPS machines before the end of the decade [Dewar-87]. These experts were right. Already, in August 87, the Tartan Laboratories validated Ada compiler for 1750A took only **100 microseconds for a parameterless RV** [Hengemihle-87] and the DDC I compiler for 80x86 generated **50 instructions, for a paremeterless, no select, 75 microsecond RV.**

4.2.4.3 FIFO Service On Entry Queues

This issue was raised at the May 87 workshop on Ada real-time issues [Brosgol-87].

Calls are currently specified by the RM to be serviced in a strict FIFO order. Therefore, high priority callers may be served after lower priority callers in contradiction of the Steelman requirement that task service be FIFO within the same priority.

Recommendations: Consider an amendment to the RM specifying that higher priority tasks' call be serviced first with FIFO ordering for same priority.

4.2.4.4 Undefined Choice Of Open Alternative

This issue was raised at the May 87 workshop on Ada real-time issues [Brosgol-87].

In the same vein, the order of service of multiple open alternative is currently unspecified.

Recommendations: Consider an amendment to the RM specifying that calls to open alternative entries be serviced in an order consistent with caller's priorities.

4.2.4.5 Priority Inversion

This issue was raised at the May 87 workshop on Ada real-time issues [Brosgol-87].

With the current RV semantics, a high priority task can be blocked, waiting for a low priority server, while another task of same or higher priority is executing.

Recommendations: Consider an amendment to the RM specifying that the

called task immediately inherit the caller's priority if it is higher.

NOTE

Currently this occurs only after RV has started (RM 9.8-5).

To complicate matters a bit, the process would have to be transitive, i.e. should the server itself be waiting for a low priority task, this task would in its turn inherit the higher priority, and so on.

4.2.5 Task Activation

The following issues were raised by the development teams:

4.2.5.1 Control Over Task Activation

Task activation occurs either at elaboration time or as the consequence of the execution of an allocator. A more efficient control is needed over task activation: GREEN featured independent activation of tasks via the "initiate" reserved word.

Recommendations: A similar effect can be achieved by declaring a "start" entry for the task.

NOTE

This solution was used for the GRODY project.

4.2.5.2 Activation Bottleneck

Activation may be a serial bottleneck for multi-processors: Creation, activation, termination and synchronization are all difficult runtime problems for distributed Ada. A recent research project at NYU (Flynn) is showing promising progress toward eliminating some of these bottlenecks on specialized hardware (IBM RP3 multi-processor) [Dewar-87].

4.2.5.3 Pre-elaboration Of Program Units

Even though it is not strictly nor exclusively a task activation issue, the following ARTEWG proposal is more relevant here than in any other runtime issue Section.

"Runtime elaboration of constants and a priori known tasks is not consistent with many embedded systems' power-up and restart requirements" [ARTEWG-2].

The RM indicates that a task specification is elaborated only once, during the parent unit's elaboration, whereas the delarative part of its body is elaborated for each activation (9.3-1).

"Real-time systems have a very limited amount of time available between power-up and first required functionality... A warm restart is a frequent recovery technique for embedded systems" [ARTEWG-2].

A recent SEI study summarizes the issue in the following way:

"The consequences of missing a real-time deadline can vary from reduction of throughput, to numerical inaccuracy, to partial loss of system functionality, or even to total system collapse. Therefore, the time taken to perform system functions such as process initiation, process termination, and context switching is crucial in a real-time multi-processing system. System start-up time is also important, as is the time taken to change operating modes, to reconfigure the system after a partial failure, or to restart the system after a total failure" [Weiderman-87].

NOTE

There are several ways to make elaboration more efficient:

- Declaring as constant all constant entities allows the compiler to perform static initializations.

- Using library package initialization for complex entities, such as tables and trees, can reduce the overhead at activation time.

For example, an embedded application may feature a constant data base loaded in Read-Only Memory (ROM). It could be argued that the rules of Ada do not prevent a compiler from considering such data as "pre-elaborated" by the compiler. In fact, embedded implementations often provide utilities, pragmas, and configuration files to handle that problem.

Recommendations: Consider the following pragma for addition to the list in appendix B of the RM.

```
pragma PRE_ELABORATE (<identifier-list>);
```

"The pragma is proposed to allow the compilation system (compiler/linker) to initialize the indicated list of data structures and a priori program units. If the list is omitted, all possible entities will be pre-elaborated, and a list of those entities that

cannot be pre-elaborated will be produced" [ARTEWG-2].

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

NOTE

This proposal raises a number of issues such as resource allocation, access to Ada entities before elaboration, etc. For this reason, there may be restrictions on the contents of pre-elaborated units.

4.2.6 Task Termination

The following issues were raised by the development teams:

4.2.6.1 Problems With Abort

"Abort semantics must be clarified by the vendor".

When a task is aborted, its dependents are killed, all delays are cancelled, and callers are sent TASKING_ERROR. Abort is extremely dangerous and should be reserved for anomalous termination.

Recommendations: Since the RM does not specify whether task completion must be synchronous or asynchronous, the vendor's documentation should indicate his choice as well as the possible side effects on the runtime system.

4.2.6.2 Abortion Via Task Identifiers

The following issue was raised by ARTEWG:

"It is sometimes necessary to abort a task that is not visible... This capability partially addresses the problem of writing reusable executives and failure-recovery tasks. If such a component is reusable, it cannot have visibility of those other tasks which it manages, since these are different for each application" [ARTEWG-2].

An application could consist in a generic unit providing a watchdog task. "Each watched task could provide its ID to the watchdog at start-up time. The watched tasks would be given an access value to a variable to be updated periodically. If the watched task failed to update the variable between checks, the watchdog would abort it" [ARTEWG-2].

Recommendations: Evaluate ARTEWG's abort procedure:

```
with TASKS_IDS; use TASKS_IDS;
procedure ABORT_TASK (I : TASK_ID);
```

NOTE

This procedure, as well as many others mentioned before, could be provided in the "LOW_LEVEL_TASKING" package mentioned at the May 87 workshop on Ada real-time issues [Brosgol-87].

Please refer to [ARTEWG-2] for rationale, program example, and other detailed considerations.

4.2.6.3 An RTE Without Abort?

Abort is responsible in part for current inefficiencies in most rendezvous implementations, since the runtime system must guard against abort at each step in the RV (at all synchronization points); this seems to be traceable to a military requirement for secure systems.

At the May 87 workshop on Ada real-time issues, the suggestion was made that a simplified RTE with no abort support be provided for applications that do not need it [Brosgol-87].

4.2.6.4 Termination Of Tasks In Library Units

The RM does not require that tasks declared in library units terminate (9.4-13).

Recommendations: The conditions under which such tasks terminate should be clarified in the RM.

Pending RM modification, these conditions must be clearly documented by the vendor.

4.2.7 Interrupt Management

The interrupt management function is responsible for the handling of several classes of events:

- Software interrupts such as UNIX signals and VMS' ASTs

- Asynchronous hardware interrupts (real-time clock, I/O devices)
- Synchronous hardware interrupts (arithmetic exceptions)

The following issues were raised by the development teams:

4.2.7.1 Interrupt Latency

"Interrupt latency must be minimized."

Consider the RM way to write a device driver ISR:

```
task UART_ISR is

    entry Transmit_buffer_empty;
    for Transmit_buffer_empty use at 16#40#;

    entry Data_received;
    for Data_received use at 16#42#;

end UART_ISR;
```

It is important that the minimum of overhead be associated with an interrupt entry. Under VMS, for instance, a device driver's ISR is executed at device IPL before being dismissed.

The RM encourages such an implementation (13.5.1-5..6) but does not require it.

Recommendations: Consider an amendment to the RM **requiring** that the corresponding entry call and task (not only the accept statement) be executed at hardware priority, in a special, low latency manner, and without invoking the tasking scheduler.

For hosted environments, this class of representation clauses may not be needed.

NOTE

See below for an alternate proposal by ARTEWG.

4.2.7.2 Fast Interrupt Pragmas

The following issue was raised by ARTEWG:

The RM allows direct calls to an interrupt entry (13.5.1-7). This facility is often described as an advantage when debugging the driver since the software can call the interrupt entry, to simulate a

hardware interrupt. However, such implementation can have a severe performance penalty.

Recommendations: To alleviate the resulting inefficiencies that could prohibit the use of Ada for real-time applications, ARTEWG proposes three pragmas that should be evaluated:

- pragma INTERRUPT_TASK (KIND : <interrupt_task_kind>); This pragma establishes stringent restrictions on the task code. This is a common practice for device drivers anyway but might be found overly restrictive.

  Basically, an interrupt task has only one entry that cannot be called by the software. The task body obeys a long list of restrictions about the kind of statements used.

  "The parameter KIND indicates the exact set of restrictions that are satisfied. The possible values include SIMPLE and SIGNALLING. A particular implementation may support additional values for KIND."

- pragma TRIVIAL_ENTRY; indicates that there is no statement in the accept block.

- pragma MEDIUM_FAST_INTERRUPT_ENTRY; "indicates that the entry at hand satisfies all those restrictions that are satisfied by an interrupt task of kind SIGNALLING, except for the restriction concerning references to non-local types and objects."

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

NOTE

The evaluation process might determine that one of these pragma is sufficient in practice.

4.2.7.3 Controlling Interrupts

The following issue was raised by ARTEWG:

Unavoidably, interrupt management is highly implementation dependent. "A common format for controlling and interrogating either individually-named or level-oriented interrupts is thus desirable" [ARTEWG-2].

However, it may be argued that such functionality is below the portability level.

Recommendations: Evaluate the following generic package (to be

instantiated by the implementation; not by the user):

```
generic

  type INTERRUPT_ID is (<>);
  -- since type is discrete, level (integer) and
  -- named (enumeration) interrupt format are
  -- supported.

package INTERRUPT_MANAGEMENT is

  type INTERRUPT_LIST is array (INTERRUPT_ID) of BOOLEAN;

  procedure ENABLE (INTERRUPT : in INTERRUPT_ID);

  procedure DISABLE (INTERRUPT : in INTERRUPT_ID);

  function ENABLED return INTERRUPT_LIST;

end INTERRUPT_MANAGEMENT;
```

NOTE

Some implementations might overload procedures ENABLE and DISABLE for type INTERRUPT_LIST to handle groups of interrupts at once since saving and restoring interrupt masks is a common activity with real-time applications.

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

4.2.8 I/O Management

The I/O management supports I/O directly or by calling on the underlying OS.

The following issues were raised by the development teams:

4.2.8.1 Put-get Problem

"When a Put is immediately followed by a get, some implementations do not flush the buffer before executing the get."

Even though this is not specifically required by the RM, the I/O example (RM 14.7), clearly indicates the designer's preference.

Recommendations: Ask the vendor to provide this functionality.

4.2.8.2 I/O From Tasks

Blocking all tasks that could otherwise proceed because one task in the main unit is suspended by an I/O operation is contrary to the spirit of the RM and totally defeats tasking on a mono-CPU machine.

Recommendations: Apply pressure on the vendor to produce a runtime system, or at least provide packages, that do not defeat tasking on mono-CPU systems.

4.2.8.3 Packet I/O For Objects Of Variable Format

Please refer to issue 1 in Section 2.5 for a complete description of the problem.

Ideally, the first field of a packet data structure should be the discriminant of a variant record type. In practice, we will have to handle in Ada "assembler level" packet design that put variant indicators, or even pieces of it, wherever they seemed to fit.

Recommendations: A package should insulate the Ada code from the variable format I/O. The package body could consist in

- Representation clauses for the discriminated type, including discriminant and the appropriate sequential_IO or other package instantiation. This is the cleanest Ada solution, but it requires that the discriminant always be at the same place in the incoming stream.

- The buffering of a packet of bytes, the extraction using representation clauses and / or functions of the discriminant, followed by explicit type conversion or unchecked conversion (if a record is involved) from the buffer to the array or variant record.

- The use of assign_to_address functions such as the one used on the NCP project at GSFC.

- If all the above solutions are demonstrated to be too slow for the application; C, assembler or direct code insertion will have to be used.

4.2.8.4 I/O Of Mixed Type Objects

As an extension of the strong typing rules, I/O in Ada deals with fixed types and, in practice, fixed size quantities. In fact, the RM allows I/O for unconstrained types but this requirement has been systematically waived by the Ada Validation Office (AVO).

It is therefore necessary to build specific packages from scratch every time that variable size and type objects have to be handled, a bad case of non-reusability and non-portability.

Recommendations: A package MIXED_IO similar to the one provided by DEC might be considered for inclusion in chapter 14.

The ACVC should check that I/O for objects of unconstrained types is supported.

4.2.8.5 Direct I/O

"Direct I/O such as DMA must be possible".

This is traditionally the realm of device drivers for embedded applications. Writing in Ada a device driver for a virtual operating system such as VMS would be dealing with the wrong level of abstraction and a dangerous exercise.

Recommendations: Vendors should provide examples of use of package LOW_LEVEL_IO for the purpose of handling DMA type I/O.

For hosted environments, where device drivers loosely coupled with the underlying OS are usually reserved for such application, the idiosyncrasies with the OS should be signalled.

4.2.9 Time Management

The time management functions support package CALENDAR and the delay statement (including selective waits and timed entry calls).

The following issues were raised by the development teams:

4.2.9.1 Timer Resolution

"Timer resolution must remain in acceptable limits".

First of all, it is important to distinguish:

1. The timer clock's period (SYSTEM.TICK)
2. The smallest possible non-null value for objects of type duration (DURATION'SMALL)
3. The accuracy for fixed point type DURATION (DURATION'DELTA)

The RM does put an upper limit of 20 ms on DURATION'SMALL and recommends that, whenever possible, less than 50 microseconds

resolution be provided (9.6-4). But, the same paragraph states that "DURATION'SMALL need not correspond to SYSTEM.TICK".

In practice, SYSTEM.TICK can reach one second. Host operating system's "real-time clock" resolutions may be responsible for such unacceptable value, but implementations should make the most honest effort to work around it. Alsys PC/AT compiler, for instance, offers a binder option that provides access to the hardware's timer. The resulting 1ms resolution is much more useful than MS-DOS' 1/18 second.

In a recent embedded system study, SEI recommends that DURATION'SMALL do not exceed 100 microseconds and that SYSTEM.TICK be less or equal to 1 millisecond [Weiderman-87].

Recommendations: Consider an amendment to the RM **requiring** that an upper limit of 20 ms for SYSTEM.TICK be specified. Make this a procurement requirement for all Ada compilers.

NOTE

> That poses problems for UNIX implementations since some of the UNIX Kernel functions such as Alarm have a 1 second resolution. However, Ualarm has a 1 microsecond resolution, and the hardware timer is accessible to the RTS.

Embedded implementations should provide a SYSTEM.TICK less or equal to 1 ms.

Furthermore, on implementations that truncate hardware timer resolution, a host dependent function should be provided that "reads the clock" with a resolution better than 50 microseconds:

```
function HI_RES_CLOCK return DURATION;
```

This function seems to belong to the ARTEWG's IDLE_DELAYS package described later.

On embedded systems where hardware varies greatly, the above function or even CALENDAR.CLOCK might be implemented as a stub. In that case, extensive documentation on how to write the body and link the object code with the RTE is needed. **A more acceptable solution is to be given the choice between high and low CLOCK resolution with a binder option (Alsys PC/AT compiler).**

4.2.9.2 Clock Jitter

"Clock jitter is unspecified in the RM".

The delay statement semantics only specifies the lower bound of the actual delay. No upper bound is guaranteed. This incertitude makes some application code, such as synchronous communications, difficult

to directly express in Ada.

This is not a problem specific to Ada, no language does better, as users of JOVIAL and CMS-2 know [Kamrad-87]. Obviously, if preemption is allowed, an upper bound could only be guaranteed for the task of highest priority, or a worst case upper bound computed for a group of such tasks.

A pragma SYNCHRONOUS could be introduced to be used as follow:

```
pragma SYNCHRONOUS
delay 200*ms;    -- THIS delay Guaranteed to be 200 ms.
```

For consistency, the same semantics should hold for the other uses of the delay statement such as timed entry calls and selective waits with delay alternative. A simpler solution that fits better in the bigger picture of Ada RTEs is for a standard RTE package to provide a procedure that will directly call the scheduler.

Recommendations: Evaluate the following procedure:

```
SUSPEND_FOR (Some_absolute_time);
```

If the interaction of this routine with the RTE is clearly documented by the vendor, an upper bound on the delay can be computed by the user or by a tool.

4.2.9.3 Special Delays

The ARTEWG proposes an interesting alternative for small delays:

"There is a need for an alternate implementation of the delay statement because:

1. A delay may be required that is less than the execution-time overhead of the Ada delay implementation

2. The semantics of the standard Ada delay, which is a task synchronization point, may not be appropriate for some applications [such as nonpreemptible Sections described in 4.2.3 (processor management)].

3. A [small] delay may be required that has a known upper bound (as well as lower bound) on duration" [ARTEWG-2].

Recommendations: Evaluate the following package, proposed by ARTEWG, that would feature a procedure to implement a special delay by busy wait.

```
package IDLE_DELAYS is

  type DURATION is
```

```
        delta <implementation defined>
        range <implementation defined>;

      procedure IDLE_DELAY (D : in DURATION);
      pragma INLINE (IDLE_DELAY);

    end IDLE_DELAYS;
```

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

NOTE

For the delay to be accurate, preemption must be disabled for the duration of the busy wait.

4.2.10 Others

These are issues that could not be classified anywhere else but are important and related to runtime environments.

The following issues were raised by the development teams:

4.2.10.1 Floating Point Representation

"There may be floating point representation incompatibilities between different languages from the same vendor, on the same machine".

Please refer to issue 1 in Section 2.1 of this document for a complete description of this problem.

Note that implementation dependent pragmas, such as DEC's pragma LONG_FLOAT (D_FLOAT), must not be trusted. Another representation may be selected by the compiler if the range of precision required does not match the representation indicated.

Recommendations: Vendors must document the various representations used and the means at the user's disposal to accommodate them.

It is good practice for users to check the Ada program against the suite of regression tests that should come with the foreign package.

4.2.10.2 Fixed Point Types

The subtleties of real types such as the "true" value of the number, the hole around 0, the relational operators trap, etc., are poorly explained in the literature. Barnes probably does the "least bad" job

here. The following example from Gerald Fisher via Doug Bryan [Bryan-87] deals with the elementary properties of fixed type numbers:

```
Type Pennies is delta 0.01 range 0.0 .. 1.0; --$
P : Pennies;
One_penny : Pennies := 0.01;
....
P := One_penny * 100;    -- One buck?
```

P will have a value anywhere between 78 cents and $1.56 because the compiler will approximate our delta of 0.01 with a "better" 1/128. Therefore, the value of One_penny will lie between 1/128 (0.007812) and 2*1/128 (0.015625). Not only is the value inconsistent with our naive expectation, but it could very well raise CONSTRAINT_ERROR, since 1.56 is clearly out of range!

Ada provides a portable way to alleviate some of the problems of this kind by using a representation clause:

```
for pennies'Small use 0.01;
```

But currently, the ACVC does not enforce any of the representation clauses and such a statement could be ignored by one compiler, such as DEC ACS 1.2 which requires a power of 2, and accepted by another.

4.2.10.3 Task Identifiers

The following issue was raised by ARTEWG:

"Several RTE extensions described in the ARTEWG catalog of runtime features and options for the Ada RTE, require a means of specifying tasks as parameters to RTE subprograms" [ARTEWG-2].

NOTE

> The RM recommends the use of access types for this purpose (9.2-7). This approach was judged inadequate in practice.

Recommendations: Evaluate the following implementation defined package proposed by ARTEWG:

```
package TASK_IDS is

  type TASK_ID is private;
  NULL_TASKS : constant TASK_ID;

  generic
    type TASK_TYPE is limited private;
  function ID_OF (T : TASK_TYPE) return TASK_ID;

  function SELF return TASK_ID;
```

```
    function ENCLOSING_TASK ( LEVELS_OUT : NATURAL)
      return TASK_ID;

    function PARENT (I : TASK_ID) return TASK_ID;

    function CALLER return TASK_ID;

    function TRANSLATE (          -- ID conversion
        SUBJECT : TASK_ID;        -- Local copy
        RECIPIENT : TASK_ID)      -- For other task
      return TASK_ID;

    function CALLABLE (I : TASK_ID) return BOOLEAN;

    function TERMINATED (I : TASK_ID) return BOOLEAN;

  private

    type TASK_ID is <implementation-defined>;

    NULL_TASK : constant TASK_ID
      := <implementation-defined>;

  end TASK_IDS;
```

NOTE

One of the applications for such a package would be a multi-tasking debugger written in Ada, since the tool would have to have convenient access to all tasks.

Since all functions are simply querying a state and not changing it, the proposal seems quite safe. However, function CALLER seems to be controversial. Languages like CSP require caller and called tasks to know each other. Ada selected the dissymetric approach in which the called task has no way of knowing who is calling it [Ichbiah-79].

Please refer to [ARTEWG-2] for rationale and other detailed considerations.

4.2.10.4 Device Allocation

There is a requirement for the non-stop Space Station systems and other critical embedded applications for dynamic allocation and de-allocation of some I/O devices.

Intermediary (queuing) tasks have been proposed and RTE have been built to provide this functionality [Auty-85].

SECTION 5

MODIFICATIONS TO THE RM

One feels rather humble when tasked to propose modifications to the Ada RM, a document produced by the best minds in computer science and scrutinized all along by hundreds of experts in dozens of countries.

A major difficulty when proposing changes to the RM is the interdependency of the Ada features. This is not a criticism. Ada's syntax is consistent and harmonious because of this; but a seemingly innocent change can have unforeseen consequences for other language constructs.

A mechanism has been put in place to accommodate real and perceived needs to change the Ada language. Users make their request to the Ada Language Maintenance Panel, which makes recommendations to the Ada board which makes recommendations to the AJPO.

Therefore, the following "proposed changes" should be seen as a loose collection of issues and thoughts from practitioners to be considered for review by the experts of the Language Maintenance Panel.

RM 3.1-8 - Elaboration: See also RM 6.1-10 for static allocation.

Issue: Elaboration must occur at runtime. That poses efficiency problems as well as some practical concerns with code in read-only storage. See Section 4.2.1 of this document.

Proposed Change: Specifically address read-only storage, and optimization issues.

RM 3.2.1-18 - Object's undefined values: See also all references to erroneous programs.

Issue: The execution of a program is erroneous if it attempts to evaluate a scalar variable with an undefined value.

Proposed Change: Implementers should be encouraged to provide set-use analysis at user's request. At least, all non-initialized variables should be set to a value such that an exception would be raised at run-time if an attempt was made to evaluate the variable before

assigning a value to it.

NOTE

Some high level optimizers already perform set use analysis. Most implementations raise CONSTRAINT_ERROR when a non-initialized variable is evaluated.

RM 4.5-7 - NUMERIC ERROR: See RM 11.1-13..14 for impact elsewhere in the RM.

Issue: Implementations may raise NUMERIC_ERROR or CONSTRAINT_ERROR under identical test conditions. See Section 4.2.2 of this document.

Proposed Change: A warning should be added in the RM about the danger of relying on one or the other exception. Examples of handlers such as RM 11.4.1-11 should be modified accordingly.

RM 4.8-7 - Storage reclamation:

Issue: "An implementation may (but need not) reclaim storage occupied by an object created by an allocator, once this object has become inaccessible." See Section 4.2.1 of this document.

Proposed Change: A way must always be provided to reclaim storage for dynamic objects.

RM 6.3.2-4 - pragma INLINE:

Issue: An implementation is free to ignore pragma INLINE under a wide range of circumstances.

Proposed Change: Pragma INLINE should be obeyed unless a compelling reason, such as a recursive subprogram, prevents it. In this latter case, a diagnostic should be provided.

RM 9.4-13 - Termination of tasks in library units:

Issue: The RM does not require that tasks declared in library units terminate.

Proposed Change: The conditions under which such tasks terminate should be clarified in the RM.

RM 9.5-15 - Entry calls FIFO order:

Issue: Entry calls are processed in FIFO order regardless of calling task's priority.

Proposed Change: Higher priority task calls should be serviced first,

with FIFO ordering for tasks of the same priority.

RM 9.6-1 - DURATION upper limit:

Issue: The upper limit for time intervals produced by the delay statement is not defined.

Proposed Change: Specifically indicate that the expiration of a delay statement is a scheduling event. Standardize an RTE procedure SUSPEND, such as the one described in Section 4.2.9 of this document.

RM 9.6-3 - Delay statement:

Issue: On some implementations, "delay 0.0;" results in a call to the scheduler. This convenient semantics should be standardized.

Proposed Change: Specify that the "delay 0.0;" semantics include the scheduling of the next executable task.

RM 9.6-4 - Type DURATION:

Issue: "DURATION'SMALL need not correspond to SYSTEM.TICK".

Proposed Change: The RM should recommend an upper limit of 20 ms for SYSTEM.TICK and a timer resolution better than 50 microseconds. See Section 4.2.9 of this document.

RM 9.8-1 - Static priorities:

Issue: The RM does not allow dynamic priorities. See Section 4.2.3 of this document.

Proposed Change: Remove the restriction that task priorities must be static and provide an example of a package LOW_LEVEL_TASKING featuring operations on type priority.

RM 9.8-1 - Default priority:

Issue: No default priority is specified.

Proposed Change: Specify that PRIORITY'FIRST is the default priority.

RM 9.8-4 - Preemptive scheduling:

Issue: The RM seems to call for preemptive scheduling, but the wording, purposely vague is subject to misunderstanding. See Section 4.2.3 of this document.

Proposed Change: More clearly disallow CPU hogging by low priority tasks. Specifically recognize the importance of predictable execution.

RM 9.8-5 - Priority during RV:

Issue: There are possible priority inversion situations because the priority inheritance occurs after RV has started.

Proposed Change: The called task should transitively inherit the caller's priority if it was higher at the time of call.

RM 13.1-11 - Bit packing for arrays of BOOLEAN:

Issue: When bit packing for arrays of BOOLEAN is not supported, separate packages or interface to assembler have to be used to perform bit manipulations. See Section 4.2.1 of this document.

Proposed Change: Clearly indicate in RM that bit packing for arrays of BOOLEAN is the official solution for bit manipulation operations.

RM 13.5.1-5..6 - Interrupt entries:

Issue: Interrupt entries may be treated like any other entries.

Proposed Change: The RM should **require** that a task with interrupt entries be executed at hardware priority without invoking the tasking scheduler.

RM 13.5.1-7 - Calls to interrupt entries:

Issue: The RM allows calls to interrupt entries. See Section 4.2.7 of this document.

Proposed Change: The RM should no longer require that such calls be allowed. Also note that the accept statement of the interrupt task and the enclosing loop statement usually enclosing it must be executed at hardware priority.

NOTE

Some implementations already do it this way.

RM 13.10.1 - Procedure UNCHECKED DEALLOCATION:

Issue: UNCHECKED_DEALLOCATION, as the RM clearly indicates, can be misused with grave consequences. See Section 4.2.1 of this document.

Proposed Change: Replace procedure UNCHECKED_DEALLOCATION by CHECKED_DEALLOCATION. Add a pragma to suppress the checks in the cases when the overhead was demonstrably intolerable.

```
pragma SUPPRESS (DEALLOCATION_CHECK, access_type_name);
```

RM B-1 - pragma PRE ELABORATE:

Issue: "Runtime elaboration of constants and a priori known tasks is not consistent with many embedded systems' power-up and restart requirements" [ARTEWG-2].

Proposed Change: A new predefined pragma:

```
pragma PRE_ELABORATE (<identifier-list>);
```

"The pragma is proposed to allow the compilation system (compiler/linker) to initialize the indicated list of data structures and a priori program units. If the list is omitted, all possible entities will be pre-elaborated, and a list of those entities that cannot be pre-elaborated will be produced" [ARTEWG-2].

A note about Chapter 13

If adopted after evaluation, ARTEWG proposed RTE packages should be standardized by including their specification in either Chapter 13 (preferably) or appendix F.

A note about RTE packages

An interesting suggestion proposed by ARTEWG's current president is to remove Chapter 13 from the RM and make a separate document of all the implementation dependent features and packages [Kamrad-87].

SECTION 6

RECOMMENDED ADA PROJECTS

Following Dr. McKay's classification of research and development activities [McKay-85], we distinguish three categories of projects that will accelerate the transition to Ada within NASA and among the Space Station contractors:

1. Studies and proof of concept for technologies that will be needed in the next decades (the "edge of the art").

2. Pilot projects and limited scale developments to gradually introduce the new technology (the state of the art).

3. Full production development making use of the proven technologies (the state of the practice).

6.1 PROOF OF CONCEPT

Even though several validated compilers are available for multiprocessor systems (Flex-32, Alliant, Sequent), research and proof of concept are sorely needed for fully distributed runtime systems, fault tolerance, and multi-level security applications [ARTEWG-6].

Testbeds such as Johnson's DMS, Lewis' ACSS, Kennedy's CDS and CSDL's AIPS should be multiplied. Most importantly, definition studies and testbeds should be funded for a Portable Common Executive Environment (PCEE) [McKay-7-87].

"Hard" real-time applications, that can fail when the timeline is not met, require that compiler generated RTEs compare in efficiency with traditionally hand coded special executives. Technologies to specify and (semi)automatically tailor runtime systems are needed [ARTEWG-6] and their proof of concept must be funded.

But even before proof of concept projects can be started, detailed studies are needed to identify and prioritize NASA's common Ada runtime environment requirements and compare them to what is available from the vendors [ARTEWG-6]. Proposals for RM changes, procurement issues, workaround, and proposed support packages will logically

follow. We hope that this study will be one of the first steps in that direction.

It is probable that three lists will emerge:

- o Short term requirements for hosted applications
- o Short term requirements for embedded and "hard" real-time applications
- o Long term requirements for all applications (distributed, fault tolerant, and multi-level security runtime systems).

When the prioritized lists of common requirements are agreed upon, consultation (perhaps in the form of a workshop) should be organized with compiler implementers to define and more precisely assess the needed work.

NOTE

> Senior representatives of two major compiler vendors were informally contacted recently and both agreed in principle with the idea of a workshop.

After negotiation, proof of concept projects to test the RTE functionality should be funded in cooperation with the compiler vendors.

One important step would be the prioritization of the issues raised in this document, a selection of some of them for implementation, and a proof of concept of some of the packages recommended. A workshop involving main contributors to this study might be a good format for deriving such a list.

6.2 PILOT PROJECTS

Probably the best way to transition an organization to Ada is to have the staff go through the following sequence:

1. Re-implementation of an existing small program (2k to 10k LOC)
2. Design and implementation of a meaningful pilot project (5k to 20k LOC)
3. First production project

The same gradual approach can be used to tackle "hard" real-time projects or new technologies such as Artificial Intelligence applications.

Most of the projects described in Sections 2 and 3 of this document

belong to the first two categories. The staff of the GRODY project at GSFC started directly from category 2.

By browsing through Sections 2 and 3 of this document, it should be relatively easy for any organization to identify pilot and first production projects.

Another meaningful pilot project could consist in the identification, design and coding of package specifications for a set of application specific software components.

6.3 PRODUCTION SOFTWARE

In spite of the extensive list of runtime issues given in this document, a large number of software projects can and should be implemented in Ada today. In fact, all but the most demanding "hard" real-time and system programming tasks can be handled with most Ada compilers, including a great variety of "soft" real-time projects. For instance, it is possible to:

- o Build reusable Ada software components for a specific application:
  - NASCOM interface
  - Decommutation
  - STOL processing
  - Communication protocols and ISDN
  - User interface
  - Trajectory computation (see JPL's projects in Section 3)
  - Simulation
  - Etc.
- o Build packages making use of the above components to provide standard interfaces at a higher level of abstraction such as:
  - ISO model
  - X-Window
  - Etc.

- Start using Ada now on non-real-time OCC functions such as OBC dump verification, STOL processing, pass initialization and "slow" display processing.

- Start experimenting with Ada for real-time applications such as

  - Re-implementing in Ada some existing time critical FORTRAN code (decommutation comes to mind)

  - Off-loading old CPUs with Ada code running on a workstation such as MicroVax II, Sun, Apollo, PC AT, Compaq 386, etc.

- Use Ada now to design and implement a new generation OCC.

- Use Ada now for embedded flight software.

- Use Ada now for OBC simulation and test support software.

- Use Ada for robotics applications.

Also, an excellent first step in the transition is to use Ada as a PDL, no matter what language is selected for implementation.

6.4 CONCLUSION

The Ada technology has evolved from the first inefficient compilers to production environments, methodologies, tools and products that are quite impressive, only four years after the adoption of ANSI-MIL standard 1815A.

By August 1987, over 100 compilers had been validated worldwide. For VAXes, there were already more Ada compilers (50 of them validated, 40 under VMS) than for any other language on any machine.

For embedded applications, truly efficient runtime systems are already available. Several occupy less than 2 kbytes. For instance, Tartan Lab's 1750A RTE requires less than 1 kbyte without tasking or access types; its full blown RTE occupies less than 10 kbytes.

RendezVous overhead significantly decreases with every release of most compilers. Already several RTEs for embedded applications feature an RV overhead under 100 instructions. For instance, DDC's compiler synchronization RV overhead for the 80186 is about 50 instructions, 70 microseconds for an 8 Mhz 80186 with 0 wait state.

Alsys' "lattice algebra" high level optimizer that eliminates nearly all unnecessary runtime checks and borders on a static debugging tool, demonstrates the great strides in practical compiler technology motivated by the Ada effort.

These are achievements the critics had said would never be seen.

Standardization such as CAIS, research such as UHCL's System Interface Set, user's group contributions such as SIGAda's, and technology transfer activities such as SEI's constitute a healthy, comprehensive and unprecedented effort.

But clearly, the current Ada Compiler Validation Capability (ACVC) is insufficient to guarantee that a validated compiler is really useful. The ACVC must be extended to enforce the entire RM, including Chapter 13 and all predefined pragmas. The current effort toward building an Ada Compiler Evaluation Capability must be accelerated and involve as large a segment of the Ada user community as is practically feasible. The AJPO must now shift its emphasis from simple validation to compiler quality assurance. The quality of the implementation including compile speed, support packages, error messages, and runtime efficiency is of great importance to the user community.

In mid-August 1987 only a few hosted Ada runtime systems were as efficient (within 10 or 20% in execution time) as their C or FORTRAN counterpart. Only one, Rational R-1000, was truly efficient across the board, but required specialized hardware. The performance of Ada tasking, already better than non-portable alternatives on some implementations, was showing signs of a breakthrough.

In mid-August 1987 over 20 compilers for embedded systems had been validated, most of them with RTEs of good performance but insufficient functionality. For instance, when Boeing compared Ada with Pascal compilers currently used for avionics software, most Ada compilers were found to produce more efficient code (2.5 times faster than Pascal) for the same memory usage [Pflug-87]. However, for embedded applications, DoD seems to emphasize the 1750A, a 16-bit limited architecture made obsolete nearly 10 years ago. The embedded systems technology still has a long way to go and needs a better direction.

After 12 years of effort (HOLWG:1975, GREEN: 1979, Ada: 1983), Ada can be credited for significantly advancing the field of practical software engineering. But in spite of thousands of comments from 15 countries and hundreds of studies and pilot projects world-wide, there is still some risk (and much-enjoyed foot-dragging) in using Ada on some systems. On most, the risk is manageable [Basili-87]. On VAX/VMS, for instance, the Ada risk is small when compared to the entire system risk. Clearly, risks are present with any software system, using any language. A significant part of these risks stem from insufficient software engineering education, an issue that is not specific to Ada.

Most importantly, the fear of change is no substitute for technical risk assessment. For nearly all non-time-critical, long lived applications, developed on good compilers, Ada is the least risky alternative today.

For time-critical applications, the Ada compilers must be tested and their runtime systems must be benchmarked and carefully compared.

Other languages' runtime systems should be benchmarked and compared in the same way to rationally assess and compare the risks. When this is done, Ada could be found to present the lowest risk [Pflug-87].

The far-sighted, consistent Ada efforts that were started at Goddard three years ago have borne fruit. In spite of shaky first compilers, experimental methodologies and insufficient expertise, several teams have rapidly demonstrated adequate proficiency in a language known for its complexity.

We conclude that, at worst, the development teams using a "sequential subset" of Ada today on good implementations, such as DEC's ACS for VAX/VMS, Alsys' for PC/AT, Apollo, and Sun, Rational for R-1000, etc., would quickly achieve productivity and runtime efficiency results comparable to those obtained by teams using FORTRAN or C. After all, FORTRAN never had tasking or memory allocation, and many difficult to maintain real-time systems have been and are still being built, behind schedule and over budget, in that 30 year old language.

Nearly two years after Ada was adopted for the Space Station only a few projects, involving production of operational Ada code are active within NASA, most of them at the Goddard Space Flight Center.

This slow progress, in spite of a clear mandate [Hall-87], points to an urgent need for a unified commitment to Ada by all NASA managers involved in Space Station development.

If NASA personnel are to control the quality and timeliness of the Ada software that will be delivered for the Space Station project, they will have to be proficient in Ada. Since it can take years to gather significant Ada expertise, more Ada activities are necessary within NASA immediately.

Ada user's group meetings, mini-conferences, and other technical gatherings such as those organized at UHCL/JSC and GSFC should be set up in other Centers as well. Presentation material should be archived and a small abstract with keyword information kept on-line to foster the sharing of experience. UHCL's data base of Ada projects, and SEL's growing library of Ada reports are steps in that direction. SSE and TMIS should provide this capability as early as possible.

Ada software must be cataloged, publicized, and its re-use systematically encouraged. The recent creation of JSC's Ada software repository is a welcome development. A separate organization partly modeled on SEI and COSMIC might be necessary to introduce and foster the use of Ada within NASA.

The Air Force has put in place Ada Insertion Offices [Klucas-87]. SEI has issued a document that details the steps necessary to introduce Ada in an organization [Foreman-87]. SEL has been gathering valuable data on meaningful projects [Godfrey-87]. All this experience is directly applicable to the Space Station Program.

In any case, Ada education and technical support should be provided to more aggressively promote the Ada technology within all NASA Centers.

In particular, it is imperative that motivated and knowledgeable managers be identified in all Space Centers as "Ada focal points" to clearly show the Agency's commitment to the Ada technology and to foster progress in its introduction.

Most of the above recommendations have already been implemented at the Goddard Space Flight Center where the introduction of Ada has been very successful on a number of projects.

APPENDIX A

BIBLIOGRAPHY

[Alger-86] Linda S. Alger and Gregory L. Greeley, "A Highly-Reliable Architecture for Executing N-Version Programming", proceedings of the 18th Joint Services Data Exchange for Inertial Systems, San Diego, California, October 30, 1986.

[Archer-86] James E. Archer et. al, "Rational's Experience Using Ada for Very Large Systems", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Armitage-85] James W. Armitage and James V. Chelini, "Ada Software on Distributed Targets: A Survey of Approaches", Ada Letters, Vol. IV, Nr. 4, January-February 1985.

[ARTEWG-1] "Catalogue of Ada Runtime Implementation Dependencies", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, November 5, 1986.

[ARTEWG-2] "A Catalog of Interface Features and Options for the Ada Runtime Environment", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, October 1986.

[ARTEWG-3] "First Annual Survey of Mission Critical Application requirements", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group.

[ARTEWG-4] "Guidelines for Effectively Using Ada Runtime Environments", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group.

[ARTEWG-5] "A Framework For Describing Ada Runtime Environments", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, November 13, 1986.

[ARTEWG-6] "The Challenge of Ada Runtime Environments, a White Paper by the ARTEWG", Association for Computing Machinery, Special Interest Group on Ada, Ada Runtime Environment Working Group, August 1987.

[Auty-85] David P. Auty, Abby Greenbaum, "Ada/M(44) Design Issues: Interrupts and I/O", AIAA / ACM / NASA / IEEE Computers in Aerospace V Conference, October 21-23, 1985.

[Auty-87] David P. Auty, "Establishing an Ada Runtime Benchmarking Capability for NASA Johnson Space Center", Softech Inc., WO-086 Vol. I to III., February 1987.

[Barnes-87] J. G. P. Barnes, "International Workshop on Real-Time Ada Issues.", Internal memorandum, Alsys Inc., July 1987.

[Basili-87] Victor Basili, et al., "Use of Ada for FAA's Advanced Automation System (AAS)", MITRE Document MTR-87W77, April 1987.

[Bass-84] Mich Bassman, "Reusable Software, Transportability and the Ada Runtime Environment", presentation at the 23rd Annual Technical Symposium of DC Chapter of the ACM, June 28, 1984.

[Becker-87] Jeffrey Becker and Robert Goettge, "Ada Performance Issues for Real-time systems", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Beser-85] Eric L. Beser, "The Westinghouse Ada Experience", Presentation at the GSFC Ada User's Group, December 1985.

[Blumberg-86] F. C. Blumberg, et. al, "Transportability, Distributability, and Rehosting Experience With a Kernel Operating System Interface Set", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Boeing-87] Robert L. Dryden (President, Boeing Computer Services), Wecoming Address at the SIGAda summer meeting, Seattle, August 26, 1987.

[Bray-83] Gary Bray, "Implementation Implications of Ada Generics", Ada Letters, Vol. III, Nr. 2, September-October 1983.

[Brennan-86] Peter Brennan, et. al, "A Distributed Environment for Ada", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Brinker-85] Elisabeth Brinker, "Comparison of Output From Source Code Compiled on Six Different Ada Compilers", NASA GSFC, Code 522, Internal Memo, May 23 1985.

[Brinker-86] Elisabeth Brinker, Curtis Emerson, Peter Hughes, "Preliminary Report On Ada Real-time Evaluation", NASA GSFC, Code 522, Internal Memo, August 28, 1986.

[Brosgol-87] Benjamin Brosgol, "International Workshop on Real-Time Ada Issues. Summary report.", Internal memorandum, Alsys Inc., June 1987.

[Bryan-87] Doug Bryan, "Dear Ada", Ada Letters, Vol. VII, Nr. 3, May-June 1987.

[Burger-87] Thomas M. Burger and Kjell W. Nielsen, "An Assessment of the Overhead Associated With Tasking Facilities and Task Paradigms in Ada", Ada Letters, Vol. VII, Nr. 1, January-February 1987.

[Burns-85] A. Burns, "Efficient Initialisations Routines for Multiprocessor Systems Programmed in Ada", Ada Letters, Vol. V, Nr. 1, July-August 1985.

[Burns-86] A. Burns, "Concurrent Programming in Ada", Cambridge University Press, 1986.

[Carlson-86] Arne Carlson, "Interesting Viewpoints to Those Who Will Put Ada Into Practice", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Carr-87] P. Carr et al., "Implementation of a Prototypr CAIS Environment", Ada Letters, Vol. VII, Nr. 2, March-April 1987.

[Cherry-86] George W. Cherry, "Process Abstraction Methodology for Embedded Large Applications (PAMELA) Handbook", Thought**Tools, Inc., Reston, Va. 1986.

[Cohen-86] Sandy Cohen, Dan McNicol, "Reusable Software Parts on a Semi-Abstract Data Type", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Conti-87] Robert A. Conti, "Critical Runtime Design Tradeoffs in an Ada Implementation", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Cornhill-83] Dennis Cornhill, "A Survivable Distributed Computing System For Embedded Application Programs Written in Ada", Ada Letters, Vol. III, Nr. 3, November-December 1983.

[Cornhill-87] Dennis Cornhill, "The Rate Monotonic Scheduler", Presented at the International Workshop on Real-Time Ada Issues, June 1987. To be published in a special issue of Ada Letters.

[CSDL-86] "Completion of the Advanced Information Processing System", The Charles Stark Draper Laboratory, Inc., November 12, 1986.

[Dapra-84] A. Dapra et al., "Using Ada and APSE to Support Distributed Multimicroprocessor Targets", Ada Letters, Vol. III, Nr. 6, May-June 1984.

[Debest-83] X. Debest, "A User-Friendly I/O System for Ada", Ada Letters, Vol. II, Nr. 4, January-February 1983.

[Dewar-87] Robert Dewar, Presentation to the Baltimore SIGAda, May 6,

1987, Baltimore, Md.

[Dewolf-84] J. Barton Dewolf, Nancy M. Sodano, and Roy S. Whittredge, "Using Ada for a Distributed, Fault Tolerant System", Proceedings of the AIAA/IEEE 6th Digital Avionics Conference, Baltimore, Md., December 3-6, 1984.

[Eisenhauer-86] Greg Eisenhauer et. al, "Distributed Ada: Methodology, Notation, and Tools", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Emerson-87] Curtis Emerson, "Ada Features List", NASA GSFC Code 522, Internal Memo, March 1987.

[Fantechi-84] A. Fantechi, "Interfacing With Real Environments From Ada", Ada Letters, Vol. III, Nr. 6, May-June 1984.

[Feinberg-86] David A. Feinberg, "Using Ada - The Deeper Challenge", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Fisher-84] Gerry Fisher, "Universal Arithmetic Packages", Ada Letters, Vol. IV, Nr. 2, September-October 1984.

[Foreman-87] John Foreman, et al., "Ada Adoption Handbook", Software Engineering Institute, 1987.

[Godfrey-87] Sara Godfrey, et al., "Assessing the Ada Design Process and its Implications: A Case Study", Software Engineering Laboratory, SEL-87-004, GSFC, July 1987.

[Goforth-87] Andy Goforth, "Experiments in Parallel Processing using C and Ada on a Multi-Processor", Ames Research Center, October 1987.

[Greeley-86] Gregory L. Greeley, "An Ada Implementation for Fault Detection, Isolation and Reconfiguration Using a Fault Tolerant Processor", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[GSFC-1] GSFC Ada User's Group, "Ada Style Guide (Version 1.1)", GSFC Document SEL-87-002, May 1987.

[Hall-87] Dana Hall, "Space Station Project - Plans and Status", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Helmbold-85] D. Helmbold and D.C. Luckham, "Runtime Detection and Description of Deadness Errors in Ada Tasking", Ada Letters, Vol. IV, Nr. 6, May-June 1985.

[Holladay] Wendy Holladay, "NSTL/ERL Space Station Payload Simulator", Internal memorandum, NSTL, 29 May 1987.

[Hood-1-86] Philip Hood and Vinod Grover, "Designing Real-time Systems in Ada", Final Report, Softech Inc., 8 January 1986.

[Hood-6-86] Philip Hood, "Ada and the Cyclic Runtime Scheduling", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Humphrey-87] Terry Humphrey, "Space Station Advanced Development: Current Tasks in Applying Ada to Space Station at NASA - JSC", Presentation at headquarters, June 1987.

[IBM-85] O. Lui, "Asessment Of Ada for Space Station Applications Software", IBM-FSD, Houston, Tx, July 1985.

[Ichbiah-79] Jean Ichbiah et al., "Rationale for the Design of the Ada Programming Language", ACM SIGPLAN notices, Vol. 14, Nr. 6, June 1979.

[Iles-87] Iles R., "TIE Routine Definitions", Toolpack/1, Release 2.1, NAG publication: NP1285, 1987.

[Intermetrics-85] "Justification for Selection of a Standard Language for Space Station Applications", Final Report, 28 June 1985.

[Invevardi-83] P. Invevardi et al., "A Distributed KAPSE Architecture", Ada Letters, Vol. III, Nr. 2, September-October 1983.

[Johnson-86] Charles S. Johnson, "Some Design constraints Required for the use of Generic Software in Embedded Systems: Packages which Manage Abstract Dynamic Structures Without the Need for Garbage Collection.", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Jouchoux-87] A. Jouchoux et al., "Developing a Spacecraft Monitor and Control System in Ada", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Kamrad-83] J. Michael Kamrad II, "Runtime Organization for the Ada Language System Programs", Ada Letters, Vol. III, Nr. 3, November-December 1983.

[Kamrad-87] J. Michael Kamrad II, "Trip Report on Ada Real Time Issues Workshop and Ada in Sweden", Honeywell Interoffice memorandum, 18 June 1987.

[Kirch-83] Walter Kirchgassner, et al., "Optimization in Ada", Ada Letters, Vol. III, Nr. 3, November-December 1983.

[Klucas-87] Col. Casper H. Klucas, "Policy Committee Session II presentation", SIGAda summer meeting, Seattle, August 25-28, 1987.

[Klumpp-86] Allan R. Klumpp, "Ada Problems and solutions", Letter to Dr. Robert Mathis, JPL, Revised June 30, 1987.

[Klumpp-87] Allan R. Klumpp, "A Collection of General-Purpose Ada Packages", JPL, Interoffice Memorandum, 314.1-0172-ARK, Revised 5 May, 1987.

[Knight-84] John C. Knight and John I. A. Urquhart, "On the Implementation and Use of Ada on Fault-Tolerant Distributed Systems", Ada Letters, Vol. IV, Nr. 3, November-December 1984.

[Knight-87] John C. Knight and Marc E. Rouleau, "Analysis of Ada for a Crucial Distributed Application", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Kok-84] J. Kok and G. T. Symm, "A Proposal for Standard Basic Functions in Ada", Ada Letters, Vol. IV, Nr. 3, November-December 1984.

[Kurbel-86] Karl Kurbel and Wolfram Pietsch, "A Portable Ada Implementation of Index Sequential Input-Output", Part 1: Ada Letters, Vol. VI, Nr. 2, March-April 1986. Part 2: Ada Letters, Vol. VI, Nr. 3, May-June 1986.

[Laird-86] James D. Laird, et. al, "Implemenation of an Ada Real-time Executive - A Case Study", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Lekkos-86] Anthony Lekkos, "DEC Ada Interface to Screen Management Guidelines (SMG)", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Lyons-86] T. G. Lyons, J. C. D. Nissen, "Selecting an Ada Environment", Cambridge University Press, 1986.

[McDonnel-85] "Space Station Data Systems Analysis / Architecture Study: Task 2 - Options Development", DR-5, Vol. 1 - Technology Options. McDonnel Douglas Astronautics Co., MDC H1940, May 1985.

[McKay-85] Dr. Charles W. McKay, Presentation to the Ada User's Group, Goddard Space Flight Center, March 1985.

[McKay-6-87] Dr. Charles W. McKay, "Lifecycle Support For 'Computer Systems' and Software Safety in the Target and Integration Environments of the Space Station Program", Software Engineering Research Center, University of Houston at Clear Lake, June 1987.

[McKay-7-87] Dr. Charles W. McKay, "Final Report on a Study of System Interface Sets For the Host, Target, and Integration Environments of the Space Station Program", Software Engineering Research Center, University of Houston at Clear Lake, July 1987.

[Maresca-86] Paul Maresca, "OASIS IBM PC/AT Rehosting Feasibility Study", Adasoft Inc., October 30, 1986.

[Martin-86] Donald G. Martin, "Non-Ada to Ada Conversion", Ada Letters, Vol. VI, Nr. 1, January-February 1986.

[Maule-86] Ruth Maule, "Runtime Implementation Issues for Real-time Embedded Ada", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Monteiro-86] Edward J. Monteiro, "Space Station Ada Runtime Support for Nested Atomic Actions", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Naedel-86] Dick Naedel, "Real-time Ada in a MC68XXX System", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Nagle-86] Gail A. Nagle, "An Ada Implementation of the Network Manager for the Advanced Information Processing System", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Naeini-86] Ray Naeini, "A Multicomputer and Real-time Ada Environment", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Nelson-85] Robert W. Nelson, "Introduction and Use of Ada at Goddard Space Flight Center", Code 522.1, MODS directorate presentation, October 29, 1985.

[Nelson-86] Robert W. Nelson, "NASA Ada Experiment -- Attitude Dynamic Simulator", Washington Ada Symposium, March 1986.

[Nissen-83] J. C. D. Nissen and B. A. Wichmann, "Ada-Europe Guidelines for Ada Compiler Specification and Selection", Ada Letters, Vol. III, Nr. 5, March-April 1984.

[Nissen-86] J. C. D. Nissen and T. G. L. Lyons, "Selecting an Ada Environment", Cambridge University Press, December 1986.

[Paulk-86] Mark C. Paulk, "Comparing Host and Target Environments for Distributed Ada Programs", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Pflug-87] Bryan Pflug, "Ada, Airplanes and Attitudes", Boeing Commercial Airplane Company, presentation at the SIGAda summer meeting, Seattle, August 25-28, 1987.

[Phillips-84] Stephen P. Phillips and Peter R. Stevenson, "The Role of Ada in Real-Time Embedded Applications", Ada Letters, Vol. III, Nr. 4, January-February 1984.

[PIWG-87] Jon S. Squire, "Ada Faster Than a Speeding Bullet, by Measurementman", PIWG presentation, SIGAda summer meeting, Seattle, August 25-28, 1987.

[Engemihle-87] Jerry Engemihle, "Explorer Platform Coprocessor Flight Software Ada Evaluation Study", Fairchild Space Company Technical Report, June 3, 1987.

[Racine-86] Roger Racine, "Transparent Rendezvous in a Fault Tolerant Distributed System", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Rockwell-83] Jody Steinbacher, Patrick Rogers, "High Order Language Study for Rockwell International Space Station Support", May 1983.

[Rogers-86] M. W. Rogers, "Ada: Language, Compilers and Bibliography", Cambridge University Press, 1986.

[RogersP-86] Patrick Rogers, Charles W. McKay, "Distributing Program Entities in Ada", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[RogersP-86-2] Patrick Rogers, "Real-time Ada", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Rudisin-87] Jerry Rudisin, "A High-Level Optimizer for Ada", presentation at the SIGAda summer meeting, Seattle, August 25-28, 1987.

[Ryer-86] Mike Ryer, "Optimization in Ada: Why it is hard", Presentation to the Washington DC SIGAda, 25 June 1986.

[Shuler-87] Robert Shuler, "Integrating Artificial Intelligence Programming Techniques with an Ada Environment", Presentation to code FR4, Kennedy Space Center.

[Spiegel-87] James R. Spiegel, "Interactive Discrete Event Simulation in Ada", Proceedings of the Joint Ada Conference (Fourth Washington Ada Symposium), Arlington, Va., March 16-19, 1987.

[Squire-87] Jon Squire (for Gil Myers), "Numerics Working Group Status Report", SIGAda summer meeting, Seattle, August 25-28, 1987.

[Taft-86] S. Tucker Taft, "A Distributed APSE", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Tasaki-87] Keiji Tasaki, J. Page, and Frank McGarry, "A Second Experiment With Ada - The GOES-I Dynamics Simulator", GSFC, SEL Project plan, May 1987.

[Tedd-84] Mike Tedd, Stefano Crespi-Reghizzi, Antonio Natali, "Ada For Multi-microprocessors", Cambridge University Press, 1986.

[Tomayko-86] James E. Tomayko, "Lessons Learned in Creating Spacecraft Computer Systems: Implications for Using Ada for the Space Station", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Vidale-86] R. F. Vidale, et. al, "Visualization, Design, and Verification of Ada Tasking Using Timing Diagrams", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Wallace-86] Robert J. Wallace, "Constructing a Working Taxonomy of Functional Ada Software Components for Real-time Embedded Applications", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

[Weiderman-87] Nelson H. Weiderman, et al., "Ada Embedded Systems: Issues and Questions", Software Engineering Institute, SEI-87-MR-2, January 1987.

[Wellings-84] A. J. Wellings et. al, "A Problem With Ada and Resource Allocation", Ada Letters, Vol. III, Nr. 4, January-February 1984.

[Whitredge-87] Roy Whitredge, "GPC Real-Time Operating System", presentation at the Charles Stark Draper Laboratory, January 21, 1987.

[Whoolley-87] Stanley Woolley, "NSTL's Ada Lessons Learned", Internal memorandum, NSTL, 19 March 1987.

[Wichmann-86] B A Wichmann, "Ackermann's Function in Ada", Ada Letters, Vol. VI, Nr. 3, May-June 1986.

[Wilder-83] William, L. Wilder, "Minimal Host for the KAPSE", Ada Letters, Vol. III, Nr. 2, September-October 1983.

[Wilder-85] William, L. Wilder, "KAPSE Implementation Strategies", Ada Letters, Vol. V, Nr. 1, July-August 1985.

[Wilke-86] Randy Wilke and Daniel Roy, "A Small Evaluation Suite For Ada Compilers", Proceedings of the First International Conference on Ada for the Space Station, University of Houston at Clear Lake, June 2-5, 1986.

APPENDIX B

PROCUREMENT ISSUES

Ada has already been used in the most varied situations from embedded microprocessors to large mainframe hosts and from MIS to real-time applications. Because of this ubiquity and the designers' concern to avoid built-in obsolescence, the RM leaves some freedom to implementers for accommodating very different needs as well as new technologies.

This appendix deals with such implementation dependencies by going through the issues raised in section 4 and making compiler procurement recommendations that address them.

B.1 COMPILER SELECTION

The following documents are relevant for the selection of Ada compilers and runtime systems:

- ARTEWG has produced a list of implementation dependencies that feature many more issues [ARTEWG-1].
- SEI has published a study for embedded applications [Weiderman-87]. Procurement issues are specifically addressed in Section 4.
- Some procurement issues for distributed implementations can be found in [Tedd-84].
- An industry standard and a more inclusive list of procurement issues can be found in [Nissen-83]. Section 5.2 and Section 6 are particularly relevant to RTE. **Every vendor should provide answers to the entire [Nissen-83] questionnaire.**

Only a few of all these issues, judged most important and relevant to RTEs, have been added to the list in section 4 of this document.

Benchmarking suites are invaluable for comparing compilers in the same environment. Suites that assess the quality of the error messages and other static features are rare [Wilke-86]. At the time of this writing, the best suite for dynamic RTE evaluation is the PIWG suite

[PIWG-87] that can be procured free of charge from SIGAda. Consult the latest issue of ACM's Ada Letters for PIWG's point of contact.

B.2 RUNTIME FEATURES AND PROCUREMENT ISSUES

The structure of this section closely parallels that of Section 4 of this document. When a feature is mandatory, "must" is used in the procurement recommendation. "Should" is used to indicate a strong preference or to state a definite need.

B.2.1 Storage Management

Any implementation that allows the user to select between static and dynamic local object allocation for non-recursive subprograms deserves extra credit. An embedded application must provide for the allocation of code from library units to the target memory.

For embedded applications, subprogram CHECKED_DEALLOCATION and the associated pragma SUPPRESS should be provided (See issue 13.10.1 in section 5). For hosted implementations, and particularly for AI applications, some form of automatic memory management should be provided. An implementation that provides such functionality deserves extra credit.

Pragma INLINE must be obeyed unless a compelling reason, such as a recursive subprogram, prevents it. In this latter case, a diagnostic must be provided. Any implementation must be compliant.

An implementation must have some way to manipulate bits and groups of bits. Any implementation that provides bit packing for arrays of BOOLEAN deserves extra credit.

For embedded applications, means to control the allocation of code from library units to the target memory must be provided, preferably outside of the Ada code.

B.2.2 Exception Management

The CONSTRAINT_ERROR exception can be raised under any one of 18 different error conditions. An implementation, the RTE of which produce accurate and precise messages to help distinguish between these conditions, deserves extra credit.

On a hosted environment, including embedded application development systems, unhandled exceptions must be propagated to the RTE and produce a traceback. In particular, this applies to exceptions raised, or propagated in tasks.

B.2.3 Processor Management

An implementation must provide a default priority and clearly document its value.

A hosted implementation should feature real preemptive scheduling and time slicing option for tasks of same priority. The functionality of ARTEWG's package TIME_SLICING could be offered in a package LOW_LEVEL_TASKING. See section 4.2.3 of this document. Embedded applications could offer a cyclic scheduler as specified in [ARTEWG-2], or CMU's rate monotonic scheduler. Furthermore, embedded implementations could offer the functionality of ARTEWG's package PREEMPTION_CONTROL in a package LOW_LEVEL_TASKING. See section 4.2.3 of this document. An implementation that provides such functionality deserves extra credit.

An embedded implementation that provides the functionality of ARTEWG's package DYNAMIC_PRIORITIES deserves extra credit.

Vendors must provide full documentation of the characteristics and behavior of the scheduler(s).

All implementations should provide pragma SHARED.

B.2.4 Rendezvous Management

A symbolic multi-task debugger that allows full control of tasks at runtime is very important for testing multi-task programs. Embedded implementations must feature an even more sophisticated toolset allowing real-time execution of the target code under the control of a debugger residing in the host [Weiderman-87].

An implementation that provides SEMAPHORE, SIGNAL, and MONITOR functionality deserves extra credit.

B.2.5 Task Activation

An implementation that provides pragma PRE_ELABORATE deserves extra credit.

B.2.6 Task Termination

The conditions under which tasks declared in library units terminate must be be documented by the vendor.

Since the RM does not specify whether task completion must be synchronous or asynchronous, the vendor's documentation must indicate his choice as well as the possible side effects on the runtime system.

B.2.7 Interrupt Management

An embedded implementation that connects to an interrupt (because of a representation clause) and does not require a non-portable pragma, deserves extra credit. Note that the accept statement and the enclosing loop statement must be executed at hardware priority.

B.2.8 I/O Management

When a Put is immediately followed by a get, the output buffer must be flushed before executing the get in accordance with RM I/O example (RM 14.7).

Do not procure a compiler the RTE of which suspends all tasks that could otherwise proceed because one task is blocked on an I/O operation. An implementation that provides non-portable constructs to alleviate such deficiency is barely acceptable.

A hosted implementation should provide a package MIXED_IO similar to the one provided by DEC.

For embedded implementations, vendors should provide examples of use of package LOW_LEVEL_IO for the purpose of handling DMA type I/O.

B.2.9 Time Management

An implementation that provides an RTE procedure SUSPEND, such as the one described in section 4.2.9 of this document, deserves extra credit.

Specify that all delay statements (including "delay 0.0;") must be implemented as scheduling events.

SYSTEM.TICK should be less than or equal to 20 ms and timer resolution better than 50 microseconds. See section 4.2.9 of this document. A hosted implementation should provide such functionality. An embedded implementation must provide a SYSTEM.TICK smaller than 1 millisecond. Furthermore, a host dependent function must be provided that "reads the clock" with a resolution better than 50 microseconds. A hosted implementation that provides such functionality deserves extra credit.

B.2.10 Others

Use of non-initialized variables should be so diagnosed by the compiler or a tool. A MAPSE that provides such functionality deserves extra credit. An implementation must raise CONSTRAINT_ERROR at run time when a variable of undefined value is evaluated.

Ask the vendor to produce an annotated chapter 13 showing what features are implemented and to what degree. Most of this chapter must be implemented. Limitations are acceptable but total elimination is not. For instance, on hosted implementations, it might be acceptable to not implement address representation clauses. This is not acceptable for embedded applications. Extensive documentation of the techniques and overhead associated with the representation clauses must be provided by the vendor. An implementation that provides guidelines on the use of representation clauses deserves extra credit.

INDEX